Formal Specification for Implementing Atomic Read/Write Shared Memory in Mobile Ad Hoc Networks Using the Mobile Unity.

Reham.A.Shihata
PhD in Computer Science, EL Menoufia University –Egypt. Software Consultant
.in Egyptian Syndicate of Programmers & Scientists
anwerreham@yahoo.com

Abstract: The Geoquorum approach for implementing atomic read/write shaved memory in mobile ad hoc networks. This problem in distributed computing is revisited in the new setting provided by the emerging mobile computing technology. A simple solution tailored for use in ad hoc networks is employed as a vehicle for demonstrating the applicability of formal requirements and design strategies to the new field of mobile computing. The approach of this paper is based on well understood techniques in specification refinement, but the methodology is tailored to mobile applications and help designers address novel concerns such as logical mobility, the invocations, specific conditions constructs. The proof logic and programming notation of mobile UNITY provide the intellectual tools required to carryout this task. Also, the quorum systems are investigated in highly mobile networks in order to reduce the communication cost associated with each distributed operation.

Keywords: Formal Specification, Mobility, Mobile Ad Hoc Networks, the Quorum Systems.

1. Introduction

Formal notations led to the development of specification languages; formal verification contributed to the application of mechanical theorem proffers to program checking; and formal derivation is a class of techniques that ensure correctness by construction, has the potential to reshape the way software will be developed in the future program derivation is less costly than post factum verification, is incremental in nature, and can be applied with varying degrees of rigor in conjunction with or completely apart from program verification. More significantly, while verification is tied to analysis and support tools, program derivation deals with the very essence of the design process, the way one thinks about problems and constructs solutions [1]. An initial highly- abstract specification is gradually refined up to the point when it contains so much detail that writing a correct program becomes trivial. Program refinement uses a correct program as starting point and alters it until a new program satisfying some additional desired properties is produced. Mobile systems, in general, consist of components that may move in a physical or logical space if the components that move are hosts, the system exhibits physical mobility. If the components are code fragments, the system is said to display logical mobility, also referred to as code mobility. Code on demand, remote evaluation, and mobile agents are typical forms of code mobility. Of course, many systems entail a combination of both logical and physical mobility (as explained in our related work). The potentially very large number of independent computing units, a decoupled computing style, frequent disconnections, continuous position changes, and the location – dependent nature of the behavior and communication patterns present designers with unprecedented challenges[1][2]. While formal methods may not be ready yet to deliver complete practical systems, the complexity of the undertaking clearly can benefit enormously from the rigor associated with a precise design process, even if employed only in the design of the most critical aspects of the system. The attempt to answer the question raised earlier consists of a formal specification and derivation for our communication protocol for ad hoc mobile systems, carrying out this exercise by employing the mobile unity proof logic and programming notation. Mobile unity provides a notation for mobile system components, coordination language for expressing interactions among the components and an associated proof Logic. This highly modular extension of the UNITY model extends both the physical and logical notations to accommodate specification of and reasoning about mobile programs that exhibit dynamic reconfiguration. Ensuring the availability and the consistency of shared data is a fundamental task for several mobile network applications. For instance, nodes can share data containing configuration information, which is crucial for carrying out cooperative tasks. The shared data can be used, for example, to coordinate the duty cycle of mobile nodes to conserve energy while maintaining network connectivity. The consistency and the availability of the data plays a crucial role in that case since the loss of information regarding the sleep/awake cycle of the nodes might compromise network connectivity. The consistency and availability of the shared data is also relevant when tracking mobile objects, or in disaster relief applications where mobile nodes have to coordinate distributed tasks without the aid of a fixed communication infrastructure. This can be attained via read/write shared memory provided each node maintains a copy of data regarding the damage assessment and dynamically updates it by issuing write operations. Also in this case it is important that the data produced by the mobile nodes does not get lost, and that each node is able to retrieve the most up-to-date information. Strong data consistency guarantees have applications also to road safety, detection and avoidance of traffic accidents, or safe driving assistance [3].The atomic consistency guarantee is widely used in distributed systems because it ensures that the distributed operations (e.g., read and write operations) performed on the shared memory are ordered consistently with the natural order of their invocation and response time, and that each local copy is conforming to such an order. Intuitively, this implies that each node is able to retrieve a copy showing the last completed update, which is crucial in cooperative tasks. However, the implementation of a fault-tolerant atomic read/write shared memory represents a challenging task in highly mobile networks because of the lack of a fixed infrastructure or nodes that can serve as a backbone. In fact, it is

DOL:10.5121/ijcsit.2022.14301

hard to ensure that each update reaches a subset of nodes that is sufficiently large to be retrieved by any node and at any time, if nodes move along unknown paths and at high speed. The focal point model provides a first answer to this challenge since it masks the dynamic nature of mobile ad hoc networks by a static model. More precisely, it associates mobile nodes to fixed geographical locations called focal points. According to this model, a focal point is active at some point in time if its geographical location contains at least one active mobile node. As a result, a focal point becomes faulty when each mobile node populating that sub-region leaves it or crashes. The merit of this model is to study node mobility in terms of failures of stationary abstract points, and to design coordination protocols for mobile networks in terms of static abstract nodes [1] [4].

2. Methodology and Notation of Mobile Unity

This section provides a gentle introduction to Mobile UNITY. A significant body of published work is available for the reader interested in a more detailed understanding of the model and its applications to the specification and verification of Mobile IP [5], and to the modeling and verification of mobile code, among others. Each UNITY .program comprises a declare, always, initially, and assign section. The declare section contains a set of variables that will be used by the program. Each is given a name and a type. The always section contains definitions that may be used for convenience in the remainder of the program or in proofs. The initially section contains a set of state predicates which must be true of the program before execution begins. Finally, the assign section contains a set of assignment statements. In each section, the symbol is"□ is used to separate the individual elements (declarations, definitions, predicates, or statements). Each assignment statement is of the form $\overrightarrow{x} = \overrightarrow{e}$ if p, where $\overrightarrow{x}$ is a list of program variables, $\overrightarrow{e}$ is a list of expressions, and $\overrightarrow{p}$ is a state predicate called the *guard* [5]. When a statement is selected, if the guard is satisfied, the right-hand side expressions are evaluated in the current state and the resulting values are stored in the variables on the left-hand side. The standard UNITY execution model involves a non-deterministic, weakly-fair execution of the statements in the **assign** section. The execution of a program starts in a state satisfying the constraints imposed by the **initially** section. At each **step,** one of the assignment statements is selected and executed. The selection of the statements is arbitrary but weakly fair, i.e., each statement is selected infinitely often in an infinite execution [5] [6].All executions are infinite. The Mobile UNITY execution model is slightly different, due to the presence of several new kinds of statements, e.g., the *reactive* statement and the *inhibit* statement described later. A toy example of a Mobile UNITY program is shown below.

Program *host (i)* **at** λ
Declare
Token: ***integer***
Initially
Token = 0
Assign
Count *token: = token* + 1
□ *Move:: λ: =Move (i, λ)*

End host

The name of the program is *host,* and instances are indexed by i. The first assignment statement in *host* increases the token count by one. The second statement models movement of the ***host*** from one location to another. In Mobile UNITY, **movement** is reduced to value assignment of a special variable λ that denotes the location of the host. We *use Move (i, λ)* to denote some expression that captures the motion patterns of *host* (i) [6].

The overall behavior of this toy example host is to count tokens while **moving.** The program *host (i)* actually defines a class of programs parameterized by the identifier i. To create a complete system, we must create instances of this program. As shown below, the **Components** section of the Mobile UNITY program accomplishes this. In our example we create two hosts and place them at initial locations λ_0 and λ_1.

System ad-hoc network
Program *host (i)* at λ
……………
End host
Components
host (0) at λ_0
□ host (1) at λ_1

Interactions
host (0).token, host (1).token:=host (0).token, host (1).token, 0
When (host (0) λ = host (1).token. λ)
^ (host (1).token$\neq$ **0**)
Inhibit *host (l).move* and *host (0).move*
When *(host (0). λ* = host (1). λ)
^ *(host (l).token > 10)*
End **ad-hoc network**

Unlike UNITY, in Mobile UNITY all variables are local to each component. A separate section specifies coordination among components by defining when and how they share data. In mobile systems, coordination is typically location dependent. Furthermore, in order to define the coordination rules, statements in the **Interactions** section can refer to variables belonging to the components themselves using a dot notation. The section may be viewed as a model of physical reality (e.g., communication takes place only when hosts are within a certain range) or as a specification for desired system services. The operational semantics of the **inhibit** construct is to strengthen the guard of the affected statements whenever the **when** clause is true. The statements in the **Interactions** section are selected for execution in the same way as those in the component programs. Thus, without the **inhibit** statement, *host(0)* and *host(l)* may move away from each other before the token collection takes place, i.e., before the first interaction statement is selected for execution. With the addition of the **inhibit** statement, when two hosts are co-located, and *host(l)* holds more than ten tokens, both hosts are prohibited from moving, until *host(l)* has fewer than eleven tokens[7]. The **inhibit** construct adds both flexibility and control over the program execution.In addition to its programming notation, Mobile UNITY also provides a proof logic, a specialization of temporal logic. As in UNITY, safety properties specify that certain state transitions are not possible, while progress properties specify that certain actions will eventually take place. The safety properties include **unless, invariant,** and **stable [7]:**

• **p unless *q*** asserts that if the program reaches a state in which the predicate (p $\wedge$ ¬*q)* holds, *p* will continue to hold at least as long as *q* does not, which may be forever.

• **Stable *p*** is defined as *p* **unless** *false,* which states that once p holds; it will continue to hold forever.

• **Inv *p*** means ((**INIT)** $\Longrightarrow$ ***p*) ^ stable *p*),** i.e., ***p*** holds initially and throughout the execution of the program. **INIT** characterizes the program's initial state.

The basic progress properties include **ensures, leads-to**, **until**, and **detects**:

• ***p* ensures *q*** simply states that if the program reaches a state where *p* is *true, p* remains *true* as long as *q* is *false,* and there is one statement that, if selected, is guaranteed to make the predicate *q true-* This is used to define the most basic progress property of programs.

• ***p* leads-to *q*** states that if program reaches a state where *p* is true, it will eventually reach a state in which *q* is true. Notice that in the **leads-to**, *p* is not required to hold until *q* is established.

• ***p* until *q*** defined as ((p **leads-to** *q)* ^ (p **unless** *q*)), is used to describe a progress condition which requires *p* to hold up to the point when *q* is established.

• ***p* detects *q*** is defined as (p $\Longrightarrow$ *q) ^ (q* **leads-to** *p)*

All of the predicate relations defined above represent a short-hand notation for expressions involving Hoare triples quantified over the set of statements in the system. Mobile UNITY and UNITY logic share the same predicate relations. Differences become apparent only when one examines the definitions of unless and ensures and the manner *in* which they handle the new programming constructs of Mobile UNITY. Here are some properties the toy-example satisfies:

(1) *(host (0).token + host (l).token* = k)
Unless *(host (0).token + host (l).token > k)*
— The total count will not decrease

(2) *host (0).token = k* leads-to *host (0).token > k*
— The number of tokens on *host* (0) will eventually increase

In the next section we employ the Mobile UNITY proof logic to give a formal requirements definition to the geoquorum approach (the application of the paper).

3. The Geoquorum-Approach (The Application)

In this paper the Geoquorum algorithm is presented for implementing the atomic read/write in shared memory of mobile ad hoc networks. This approach is based on associating abstract atomic objects with certain geographic locations. It is assumed that the existence of Focal Points, geographic areas that are normally "populated" by mobile nodes. For example: a focal point may be a road Junction, a scenic observation point . Mobile nodes that happen to populate a focal point participate in implementing a shared atomic object, using a replicated state machine approach. These objects, which are called focal point objects, are prone to Occasional failures when the corresponding geographic areas are depopulated. The Geoquorums algorithm uses the fault-prone focal point objects to implement atomic read/write operations on a fault-tolerant virtual shared object. The Geoquorums algorithm uses a quorum- based strategy in which each quorum consists of a set of focal point objects. The quorums are used to maintain the consistency of the shared memory and to tolerate limited failures of the focal point objects, which may be caused by depopulation of the corresponding geographic areas. The mechanism for changing the set of quorums has presented, thus improving efficiency [8]. Overall, the new Geoquorums algorithm efficiently implements read/write operations in a highly dynamic, mobile network. In this chapter, a new approach to designing algorithms for mobile ad hoc networks is presented. 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, which co-operate to route communication from sources to destinations. Ad hoc communication networks are by nature, highly dynamic. Mobile nodes are often small devices with limited energy that spontaneously join and leave the network. As a mobile node moves, the set of neighbors with which at 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 using classical tools. The difficulties arise from the lack of a fixed infrastructure to serve as the backbone of the network. In this section developing a new approach that allows existing distributed algorithm to be adapted for highly dynamic ad hoc environments one such fundamental problem in distributed computing is implementing atomic read/ write shared memory [8]. Atomic memory is a basic service that facilitates the implementation of many higher level algorithms. For example: one might construct a location service by requiring each mobile node to periodically write its current location to the memory. Alternatively, a shared memory could be used to collect real – time statistics, for example: recording the number of people in a building here, a new algorithm for atomic multi writes/multi- reads memory in mobile ad hoc networks. The problem of implementing atomic read/write memory is divided into two parts; **first**, we define a static system model, the focal point object model that associates abstract objects with certain fixed geographic locales. The mobile nodes implement this model using a replicated state machine approach. In this way, the dynamic nature of the ad hoc network is masked by a static model. Moreover it should be noted that this approach can be applied to any dynamic network that has a geographic basis. **Second**, an algorithm is presented to implement read/write atomic memory using the focal point object model. The implementation of the focal point object model depends on a set of physical regions, known as focal points .The mobile nodes within a focal point cooperate to simulate a single virtual object, known as a focal point object. Each focal point supports a local broadcast service, LBcast 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 focal broadcast service is used to implement a type of replicated state machine, one that tolerates joins and leaves of mobile nodes. If a focal point becomes depopulated, then the associated focal point object fails. (Note that it doesn't matter how a focal point becomes depopulated, be it as a result of mobile nodes failing, leaving the area, going to sleep. etc. Any depopulation results in the focal point failing). The Geoquorums algorithm implements an atomic read/write memory algorithm on top of the geographic abstraction, that is, on top of the focal point object model. Nodes implementing the atomic memory use a Geocast service to communicate with the focal point objects. In order to achieve fault tolerance and availability, the algorithm replicates the read/write shared memory at a number of focal point objects. In order to maintain consistency, accessing the shared memory requires updating certain sets of focal points known as quorums. An important aspect of our approach is that the members of our quorums are focal point objects, not mobile nodes. The algorithm uses two sets of quorums (I) **get-quorums** (II) **put- quorums** with property that every get-quorum intersects every put-quorum. There is no requirement that put-quorums intersect other put-quorums, or get-quorums intersect other get-quorums. The use of quorums allows the algorithm to tolerate the failure of a limited number of focal point objects. Our algorithm uses a Global Position System (GPS) time service, allowing it to process write operations using a single phase, prior single-phase write algorithm made other strong assumptions, for example: relying either on synchrony or single writers. This algorithm guarantees that all read operations complete within two phases, but allows for some reads to be completed using a single phase: the atomic memory algorithm flags the completion of a previous read or write operation to avoid using additional phases, and propagates this information to various focal paint objects[9]. 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 times it may be desirable to use different sets of get quorums and put-quorums. For example: during intervals when there are many more read operations than write operations, it may be preferable to use smaller get- quorums that are well distributed, and larger put-quorums that are sparsely distributed. In this case a client can rapidly communicate with a get-quorum while communicating with a put – quorum may be slow. 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 quorum systems, thus changing the available put-quorums and get –quorums. As a result of the static underlying focal point object model, in which focal point objects neither join nor leave, it isn't a severe limitation to require the number of predetermined quorum systems to be finite (and small). The resulting reconfiguration algorithm, however, is quite efficient compared to prior reconfigurable atomic memory algorithms. Reconfiguration doesn't significantly delay read or write operations, and as no consensus service is required, reconfiguration terminates rapidly.

The mathematical notation for the geoquorum approach

- I the totally- ordered set of node identifiers.
- $I_0 \in I$, a distinguished node identifier in I that is smaller than all order identifiers in I.
- S, the set of port identifiers, defined as $N^{>0} \times OP \times I$,
  Where OP= {get, put, confirm, recon- done}.
- O, the totally- ordered, finite set of focal point identifiers.
- T, the set of tags defined as $R^{\geq 0} \times I$.
- U, the set of operation identifiers, defined as $R^{\geq 0} \times S$.
- X, the set of memory locations for each $x \in X$:
  - V_x the set of values for x
  - $v_{0,x} \in V_x$, the initial value of X
- M, a totally-ordered set of configuration names
- $c_0 \in M$, a distinguished configuration in M that is smaller than all other names in M.
- C, totally- ordered set of configuration identifies, as defined as: $R^{\geq 0} \times I \times M$
- L, set of locations in the plane, defined as $R \times R$

Fig .1 Notations Used in The Geoquorums Algorithm.

Variable Types for Atomic Read/Write object in Geoquorum Approach for Mobile Ad Hoc Network

The specification of a variable type for a read/write object in geoquorum approach for mobile ad hoc network is presented. A read/write object has the following variable type (see fig .2) [8].

Put/get variable type τ

State

Tag $\in$ T, initially< $0.i_0$>

Value $\in$ V, initially v_0

Config-id $\in$ C, initially< 0, i_0, c_0>

```
Confirmed-set $C \subseteq T$, initially Ø
Recon-ip, a Boolean, initially false
Operations
Put (new-tag, new-value, new-Config-id)
If (new-tag> tag) then
 Value ←new-value
Tag ← new-tag
If (new-Config-id > Config-id) then
Config-id ← new-config-id
Recon-ip   ← true
Return put-Ack (Config-id, recon-ip)
Get (new-config-id)
If (new-config-id >Config-id) then
Config-id ← new-Config-id
Recon-ip ←true
Confirmed ← (tag ∈ confirmed-set)
Return get-ack (tag, value, confirmed, Config-id, recon-ip)
Confirm (new-tag)
Confirmed-set ←confirmed –set U {new-tag}
Return confirm-Ack
Recon –done (new-Config-id)
If (new-Config-id=Config-id) then
Recon-ip ←false
Return recon-done-Ack (   )
```

Fig .2 Definition of the Put/Get Variable Type τ

3.1 Operation Manager

In this section the Operation Manger (OM) is presented, an algorithm built on the focal/point object Model. As the focal point Object Model contains two entities, focal point objects and Mobile nodes, two specifications is presented , on for the objects and one for the application running on the mobile nodes [9] [10].

3.1.1 Operation Manager Client

This automaton receives read, write, and recon requests from clients and manages quorum accesses to implement these operations (see fig .3). The Operation Manager (OM) is the collection of all the operation manager clients (OM_i, for all i in I).it is composed of the focal point objects, each of which is an atomic object with the put/get variable type:

Operation Manager Client Transitions

```
Input write (Val) $_i$
Effect:
Current-port-number ⟵
Current-port-number +1
Op ⟵        < write, put, <clock, i>, Val, recon-ip, <0, i0, c0>, Ø>
Output write-Ack ( ) $_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
If op .recon-ip then
$\surd$ $C' \in M$, э $P \in$ put-quorums($C'$): $P \subseteq$ op. acc
Else
Э $P \in$ put-quorums(C): $P \subseteq$ Op. acc
Op .phase=put
Op. type=write
Effect:
Op. phase ⟵        idle
Confirmed ⟵        confirmed U {op. tag}
Input read ( ) $_i$
Effect:
Current-port-number ⟵
Current-port-number +1
Op ⟵        < read, get, $\perp$, $\top$, recon-ip, <0, i0, c0>, Ø>
Output read-ack (v) $_i$
Precondition:
```

```
Conf-id=<time-stamp, Pid, c>
If op. recon-ip then
$\surd\, C' \in M$, э $G \in$ get-quorums($C'$): $G \underline{C}$ op. acc
Else
Э $G \in$ get-quorums(C): $G \underline{C}$ op. acc
Op. phase=get
Op. type=read
Op. tag $\in$ confirmed
v= op. value
Effect:
Op .phase $\longleftarrow$ idle
Internal read-2( )$_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
$\surd\, C' \in M$, э $G \in$ get-quorums($C'$): $G \underline{C}$ op. acc
Else
Э $G \in$ get-quorums(C): $G \underline{C}$ op. acc
Op. phase=get
Op. type=read
Op. tag $\notin$ confirmed
 Effect:
Current-port-number $\longleftarrow$
Current-port-number +1
Op. phase $\longleftarrow$ put
Op. Recon. ip $\longleftarrow$ recon-ip
Op. acc $\longleftarrow$ Ø
Output read-Ack (v)$_i$
Precondition:
Conf-id=<time-stamp, Pid, c>
If op. recon-ip then
$\surd\, C' \in M$, э $P \in$ put-quorums($C'$): $P \underline{C}$ op. acc
Else
Э $P \in$ put-quorums(C): $P \underline{C}$ op. acc
Op. phase=put
Op. type=read
v=op. value
Effect:
Op. phase $\longleftarrow$ idle
Confirmed $\longleftarrow$ confirmed U {op. tag}
Input recon (conf-name)$_i$
Effect:
Conf-id $\longleftarrow$ <clock, i, conf-name>
Recon-ip $\longleftarrow$ true
Current-port-number $\longleftarrow$
Current-port-number +1
Op $\longleftarrow$ < recon, get, $\perp$, $\perp$, true, conf-id, Ø>
Internal recon-2(cid) $_i$
Precondition
$\surd\, C' \in M$, э $G \in$ get-quorums($C'$): $G \underline{C}$ op. acc
$\surd\, C' \in M$, э $P \in$ put-quorums($C'$): $P \underline{C}$ op. acc
Op. type=recon
Op. phase=get
Cid=op. recon-conf-id
Effect
Current-port-number $\longleftarrow$
Current-port-number +1
Op. phase $\longleftarrow$ put
Op. acc $\longleftarrow$ Ø
Output recon-Ack(c) $_i$
Precondition
Cid=op. recon-conf-id
Cid= <time-stamp, Pid, c>
```

```
Э P ∈put-quorums(C): P C̲ op. acc
Op. type=recon
Op. phase=put
Effect:
If (conf-id=op. recon-conf-id) then
Recon-ip  ←———  false
Op. phase ←———  idle
Input geo-update (t, L) i
Effect:
Clock ←———  1
```

Fig .3 Operation Manager Client Read/Write/Recon and Geo-update Transitions for Node

3.2 Focal Point Emulator Overview

The focal point emulator implements the focal point object Model in an ad hoc mobile network. The nodes in a focal point (i.e. in the specified physical region) collaborate to implement a focal point object. They take advantage of the powerful LBcast service to implement a replicated state machine that tolerates nodes continually joining and leaving .This replicated state machine consistently maintains the state of the atomic object, ensuring that the invocations are performed in a consistent order at every mobile node [8].In this section an algorithm is presented to implement the focal point object model. the algorithm allows mobile nodes moving in and out of focal points, communicating with distributed clients through the geocast service, to implement an atomic object (with port set q=s)corresponding to a particular focal point. We refer to this algorithm as the Focal Point Emulator (FPE). The FPE client has three basic purposes. First, it ensures that each invocation receives at most one response (eliminating duplicates).Second, it abstracts away the geocast communication, providing a simple invoke/respond interface to the mobile node [9]. Third, it provides each mobile node with multiple ports to the focal point object; the number of ports depends on the atomic object being implemented. The remaining code for the FPE server is in fig .4.When a node enters the focal point, it broadcasts a join-request message using the LBcast service and waits for a response. The other nodes in the focal point respond to a join-request by sending the current state of the simulated object using the LBcast service. As an optimization, to avoid unnecessary message traffic and collisions, if a node observes that someone else has already responded to a join-request, and then it does not respond. Once a node has received the response to its join-request, then it starts participating in the simulation, by becoming active. When a node receives a Geocast message containing an operation invocation, it resends it with the Lbcast service to the focal point, thus causing the invocation to become ordered with respect to the other LBcast messages (which are join-request messages, responses to join requests, and operation invocations).since it is possible that a Geocast is received by more than one node in the focal point ,there is some bookkeeping to make sure that only one copy of the same invocation is actually processed by the nodes. There exists an optimization that if a node observes that an invocation has already been sent with LBcast service, then it does not do so. Active nodes keep track of operation invocations in the order in which they receive them over the LBcast service. Duplicates are discarded using the unique operation ids. The operations are performed on the simulated state in order. After each one, a Geocast is sent back to the invoking node with the response. Operations complete when the invoking node with the response. Operations complete when the invoking node remains in the same region as when it sent the invocation, allowing the geocast to find it. When a node leaves the focal point, it re-initializes its variables .A subtle point is to decide when a node should start collecting invocations to be applied to its replica of the object state. A node receives a snapshot of the state when it joins. However by the time the snapshot is received, it might be out of date, since there may have been some intervening messages from the LBcast service that have been received since the snapshot was sent. Therefore the joining node must record all the operation invocations that are broadcast after its join request was broadcast but before it received the snapshot .this is accomplished by having the joining node enter a "listening" state once it receives its own join request message; all invocations received when a node is in either the listening or the active state are recorded, and actual processing of the invocations can start once the node has received the snapshot and has the active status. A precondition for performing most of these actions that the node is in the relevant focal point. This property is covered in most cases by the integrity requirements of the LBcast and Geocast services, which imply that these actions only happen when the node is in the appropriate focal point [11][12].

```
Focal Point Emulator Server Transitions
Internal join ( ) Obj , i
Precondition:
Location ∈ FP-location
Status=idle
Effect:
Join-id ←<clock, i>
Status← joining
Enqueue (Lbcast-queue,<join-req, join-id>)
Input Lbcast- rcv (< join-req, jid>) obj, i
Effect:
If ((status=joining)) ^ (jid=Join-id)) then
Status ←listening
If ((status=active))) ^ jid ∉ answered-join-reqs)) then
```

```
Enqueue (LBcast-queue, < join-ack, jid, val>)
Input Lbcast- rcv (<join-ack, jid, v>) $_{obj,\ i}$
Effect:
Answered-join-reqs ← answered-join-reqs U {jid}
If ((status=listening)  ^ (jid =join-id)) then
Status ← active
val ⟵ V
Input Geocast –rcv (< invoke, inv, oid, loc, FP-loc>) $_{obj,i}$
Effect:
If (FP-loc=FP-location) then
If (<inv, oid, loc> $\notin$ pending-ops U completed ops) then
Enqueue (Lbcast-queue, <invoke, inv, oid, loc>)
Input LBcast –rcv (< invoke, inv, oid, loc>) $_{obj,i}$
Effect:
If ((status=listening V active) ^
(<inv, oid, loc> $\notin$ pending-ops U completed-ops)) Then
Enqueue (pending-ops, <inv, oid, loc>)
Internal simulate-op (inv) $_{obj,\ i}$
Precondition:
Status=active
Peek (pending-ops) =<inv, oid, loc>
Effect:
(Val, resp)← $\delta$ (inv, val)
 Enqueue (geocost- queue, < response, resp, oid, loc>)
Enqueue (completed-ops, Dequeue (pending-ops))
Internal leave (   ) $_{obj,\ i}$
Precondition:
Location $\notin$ fp-location
Status $\neq$ idle
Effect:
Status← idle
Join-id← <0, $i_0$>
Val ← $v_0$
Answered -join- reqs← Ø
Pending –ops ← Ø
Completed-ops ← Ø
Lbcast-queue ← Ø
Geocast-queue ← Ø
Output Lbcast (m) $_{obj,\ i}$
Precondition:
Peek (Lbcast-queue) =m
Effect:
Duqueue (Lbcast- queue)
Output geocast (m) $_{obj,\ i}$
Precondition:
Peek (geocast-queue) =m
Effect:
Dequeue (geocost- queue)
Input get-update (l, t) $_{obj,i}$
Effect:
Location ← l
Clock← t
```

Fig. 4 FPE server transitions for client i and object Obj of variable type τ = <V, v_0, invocations, responses, δ >

4. Mobile Quorum Systems

In this section, the quorum systems in highly mobile networks are investigated in order to reduce the communication cost associated with each distributed operation. Our analysis is driven by two main reasons: (1) guarantee data availability, and (2) reduce the amount of message transmissions, thus conserving energy. The availability of the data is strictly related to the liveness and response time of the recovery protocol since the focal point failures occur continuously, as they are triggered by the motion of nodes. Quorum systems are well-known techniques designed to enhance the performance of distributed systems, such as to reduce the access cost per operation and the load. A quorum system of a universe *U* is a set of subsets of *U*, called *quorums,* such that any pair of quorums does intersect. In this paper, the analyzing of quorum systems is in condition of high node mobility. Note that the universe U of our quorum systems is (Focal Point) FP, a set of n stationary focal points . This choice allows us to study node mobility in terms of continuous failures of stationary nodes. In the next Section, two examples of quorum systems are analyzed and show that they are not always able to guarantee data consistency and availability under the mobility constraints, and provide in Lemma 1 a condition on the size of the minimum quorum intersection that is *necessary* to guarantee these properties [13].

4.1 Quorum Systems under Mobility Model

This section shows here that quorums proposed for static networks are not able to guarantee data consistency and availability if assumptions A_1 and A_2 hold, because the minimum quorum intersection is not sufficiently large to cope with the mobility of the nodes. In fact, since read/write operations are performed over a quorum set, in order to guarantee data consistency each read quorum must intersect a quorum containing the last update. We show that there are scenarios that invalidate this condition in case of quorum systems Q_g with non-empty quorum intersection, and in case of dissemination quorum systems Q_d with minimum quorum intersection equal to $f + 1$, where f is the maximum number of failures [14].

4.1.1 Generic Quorum System

It is a set of subsets of a finite universe U such that, any two subsets *(quorums)* intersect (consistency property) and, there exists at least one subset of correct nodes (availability property). The second condition ensures data availability and poses the constraint $f < \frac{n}{2}$. In our system model where nodes continuously fail and recover, this condition is not sufficient to guarantee data availability. For instance, in an implementation of a read/write atomic memory based on Q_g, the liveness of the read protocol can be violated since it terminates only after receiving a reply from *a full quorum* of active focal point. Therefore, since the recovery operation involves a read operation, data can become unavailable [10][11].

4.1.2 Dissemination Quorum Systems

They satisfy a stronger consistency property, but insufficient if failures occur continuously. An *f-fail-prone system* $ß \subset 2^U$ of U is defined as a set of subsets of faulty nodes of U none of which is contained in another, and such that some $B \in ß$ contains all the faulty nodes (whose number does not exceed f).

Definition 1. *A dissemination quorum system Q_d of U for a f -fail-prone system ß, is set of subsets of U with the following properties:*

(i) $| Q_1 \cap Q_2 | ¢ B \surd\ Q_1 , Q_2 \in Q_{d,} \surd B \in ß$

(ii) $\surd B \in ß \ni Q \in Q_d : Q \cap B \neq Ø$.

Dissemination quorum systems tolerate less than n/3 failures. Unfortunately, since in our system model an additional focal point might fail between the invocation and the response time of a distributed operation, more than f focal points in a quorum set can be non-active at the time they receive the request. As a result, data availability can be violated. The following lemma provides a condition on the minimum quorum intersection size (lower bound) that is *necessary* to guarantee data consistency and availability under our system model, provided nodes fail and recover [15].

Lemma 1. *An implementation φ of a read/write shared memory built on top of a quorum system* δ. *of a universe* FP *of stationary nodes that fail and recover according to assumptions A_1, A_2 guarantees data availability only if* $|Q_1 \cap Q_2| > f + 1$ *for any Q_1 , $Q_2 \in Q$. It ensures atomic consistency only if* $|Q_1 \cap Q_2 > f + 2$ *for any Q_1 , $Q_2 \in Q$.*

4.2The MDQ Quorum Systems

This section introduces here a new class of quorum systems, called *Mobile Dissemination Quorum system* (MDQ) that satisfies the condition in Lemma 1.

Definition 2. A MDQ system Q_m of FP is a set of subsets of FP such that $|Q_1 \cap Q_2| > f + 2$ for any Q_1 , $Q_2 \in Q$. Note that in contrast with Q_g the liveness of the distributed operations performed over a quorum set is guaranteed by the *minimum number* of alive nodes contained in any quorum. As a result, in case of failures the sender does not need to access another quorum in order to complete the operation. This improves the response time in case of faulty nodes and reduces the message transmissions. Let us consider now the following MDQ system:

$$Q_{opt} = \left\{ Q : (Q \subset \underline{FP}) \wedge \left(|Q| = \left\lceil \frac{n + f + 3}{2} \right\rceil \right) \right\}$$

Lemma 2. *Q_{opt} is a MDQ system and $f \leq n - 3$.*

Proof: Since $|Q_1 \cup Q_2| = |Q_1| + |Q_2| - |Q_1 \cap Q_2|$ for any

$$Q_1, Q_2 \in Q_{opt,}$$

And $|Q_1 \cup Q_2| \leq n$, then $|Q_1 \cap Q_2| \geq n+f+3-n$.

In addition, Q_{opt} tolerates up to $n - 3$ failures since the size of a quorum cannot exceed n, that is

$$\left\lceil \frac{n+f+3}{2} \right\rceil \leq n$$

This implies (f+3-n) / 2 ≤ 0 note that Q_{opt} is highly resilient (in the trivial case f = n -3, $Q_{opt} = \{U\}$). Clearly, there is a trade-off between resiliency and access cost since the access cost per operation increases with the maximum number of failures. Moreover, our assumption of connectivity among active focal points becomes harder to guarantee as f becomes larger. It is important to note that the minimum intersection size between two quorums of Q_{opt} is equal to $f + 3$. We prove in the following section that there exists an implementation of atomic memory built on top of Q_{opt} This shows that $f + 3$ is the minimum quorum intersection size necessary to guarantee data consistency and data availability under our mobility model. Therefore, Q_{opt} is *optimal* in the size of the minimum quorum intersection, that is in terms of message transmissions since the sender can compute a quorum consisting of its $\left\lceil \frac{n+f+3}{2} \right\rceil$ closest nodes. This is particularly advantageous in sensor networks because it can lead to energy savings [15].

4.3 An Implementation of Read/Write Atomic Memory

In this section, the Q_{opt} is the quorum system with minimum intersection size f+3 that is able to guarantee data consistency and availability under our system model and mobility constraints. We prove that by showing that there exists an implementation φ of atomic read/write memory built on top of Q_{opt}. Our implementation consists of a suite of read, write and recovery protocols and built on top of the focal points and the Qbcast abstraction [13][14].

4.3.1 The Qbcast Service

A focal point F_i is *faulty* at time t if focal point region G_i does not contain any active node at time t or F_i is not connected to a quorum of focal points. In our implementation each read, write and recovery request is forwarded to a quorum of focal points. This task is performed by the *Qbcast* service. It is tailored for the MDQ system and designed for hiding lower level details. Similarly to Qbcast guarantees reliable delivery. It is invoked using interface *qbcast (m)*, where m is the message to transmit containing one of these request tags write, read, confirm. The notation $\{s_i\}_{i \in Q}$ *qbcast* **(***m***,** *Q* denotes the Qbcast invocation over quorum Q, $\{s_i\}_{i \in Q}$ the set of replies, where $Q \subset Q$. We call the subset Q the *reply set* associated with request m. This set plays a crucial role to prove data availability and atomic consistency. Upon receiving a request m, Qbcast computes a quorum $Q \in Q_{opt}$ and transmits message m to each focal point in Q. It is important to note that *qbcast* (*m*) returns *only if* the node receives within T time units at least $|Q| - (f + 1)$ replies from Q. If this does not occur, it waits for a random delay and retries later since if this happens the focal point is faulty by our definition. Note that if read (or write) operations occur more frequently than write (or read) operations, we can reduce message transmissions by distinguishing between read and write and making read (or write) quorums smaller. However, for simplicity of presentation we do not distinguish between read and write quorums [13] [15].

4.3.2 Protocols

The high level description of the read/write/ recovery protocols is illustrated in Fig.5. Each mobile node maintains a copy of the state s associated with the shared variable x, which is a compound object containing the value *s.val* of x, a timestamp *s.t* representing the time at which a node issued update *s.val*, and a confirmed tag that indicates if *s.val* was propagated to a quorum of focal points. Each node can issue write, read and recovery operations. A new state is generated each time a node issues.

```
Write (v):
s ← {v, t, unconfirmed, rand}
$\{ack_i\}_{i \in Q}$ ← qbcast (<write s>)
$\{ack_i\}_{i \in Q}$ ← qbcast (<confirm s>)
Read/recovery ( ):
$\{s_i\}_{i \in Q}$ ← qbcast (<read>)
s ← state $(\{s_i\})_{i \in Q}$
if (s not confirmed)
$\{ack_i\}_{i \in Q}$ ← qbcast (<confirm, s>)
return s.val
```

Fig.5 Write / Read/Recovery Protocols.

4.3.3 Write Protocol

A node *C* requesting a write *v* computes a new state *s* consisting of value *v*, the current timestamp, tag unconfirmed, and a random identification rand. It transmits its update to a quorum of focal points via the Qbcast service by invoking *qbcast (*<write, *s>)* and successively *qbcast* (< confirm, *s>)* to make sure that a quorum of focal points received such an update.Upon receiving a write request, each non-faulty focal point (including recovering) replaces its state with the new state *s* only if the associate timestamp *s.t* is higher than the timestamp of its local state, and sets its write tag to unconfirmed. This tag is set to confirmed upon receiving the confirm request sent in the second phase of the write protocol, or sent in the second phase of the read protocol in case the node that issued the write operation could not complete the write operation due to failure[13].

4.3.4 Read Protocol

In the read protocol, a node *C* invokes *qbcast* (<read>), which forwards the read request to a quorum *Q* of focal points. Each non-faulty focal point in *Q* replies by sending a copy of its local state *s*. Upon receiving a set of replies from the Qbcast service, node *C* computes the state with highest timestamp and returns the corresponding value. If the tag of *s* is equal to unconfirmed, it sends a confirm request. This is to guarantee the linearizabitity of the operations performed on the shared data in case a write operation did not complete due to client failure [11][12].

4.3.5 Recovery Protocol

It is invoked by a node C upon entering an empty region G_i. More precisely, *C* broadcasts a join request as soon as it enters a new focal point region and waits for replies. If it does not receive any reply within *2d* time units, where *d* is the maximum transmission delay, it invokes the recovery protocol which works in the same way as the read protocol.

4.4 Analysis

In this section, the key steps to prove the atomic consistency and data availability of the implementation presented in this paper is shown.

A. Data Availability

The availability of the data is a consequence of our failure model and of the Qbcast service. The following lemmas are useful to prove it and will be also used in showing atomic consistency [13].

Lemma 3. *The Qbcast service invoked by an active focal point terminates within τ time units since the invocation time.*
Proof*:* This is true since an active focal point or is able to communicate with a quorum of focal points because of Definition 2, and because at most $f + 1$ focal points in *Q* can be faulty when the request reaches their focal point regions. In fact, because of assumptions A_1 and A_2 at most $f + 1$ focal points can appear to be faulty during τ time units. Therefore, at least $|Q|$ — $(f+ 1)$ focal points in a quorum reply. This proves our thesis since the QBcast service guarantees reliable delivery, and the maximum round-trip transmission delay is equal to τ.
The following lemma and Theorem 1 is a straightforward derivation of the liveness of the Qbcast service.
Lemma 4. *An active focal point recovers within τ time units.*
Theorem 1. *This implementation of atomic read/write shared memory guarantees data availability.*
Lemma 5. *At any time in the execution there are at most f+ I faulty and recovering focal points.*

Proof. Because of Assumptions A_1 and A_2, and Lemma 4, there are at most $f + 1$ faulty and recovering focal points during any time interval $[t, t + \tau]$ for any time *t* in the execution. This can occur if there are *f* faulty focal points before *t* and during $[t, t + \tau]$ one of these faulty focal points recovers and another one fails.

B. Atomic Consistency

There exists a total ordering of the operations with certain properties. We need to show that the total order is consistent with the natural order of invocations and response. That is, if o_1 completes before o_2 begins, then $o_1 <_a o_2$.
Lemma 6. *The reply set Q associated wi<u>th</u> a request satisfies the following properties:*

$$(i)\ \left|\bar{Q}\right| \geq \left\lceil \frac{n-f}{2} \right\rceil;$$

$$(ii)\ \left|\bar{Q} \cap Q\right| \geq 2 \quad \textit{for} \quad \text{any} \quad Q \in Q_{\text{opt}}$$

Proof. The first property holds because the QBcast service completes only upon receiving at least $|Q|$ - $(f+ 1)$ replies from a quorum of servers. Therefore,

$$\left|\bar{Q}\right| \geq \frac{n-f+1}{2} \geq \left\lceil \frac{n-f}{2} \right\rceil$$

Since $|\bar{Q} U Q| = |\bar{Q}| + |\bar{Q}| - |\bar{Q} \cap Q|$ and $|QUQ| \leq$ n, then

$$\left|Q \cap \bar{Q}\right| \geq \left\lceil \frac{n-f}{2} \right\rceil + \left\lceil \frac{n+f+3}{2} \right\rceil - n$$

Therefore, since $\left\lceil \frac{a}{2} \right\rceil + \left\lceil \frac{b}{2} \right\rceil \geq \left\lceil \frac{a+b}{2} \right\rceil$ for any a, b $\in R$, then

$$\left|Q \cap \bar{Q}\right| \geq \left\lceil n + \frac{3}{2} \right\rceil - n = 2.$$

Lemma 7. *Let o_1 be a write operation with associated state s_1. Then, at any time t in the execution with t > res (o_1) there exists a subset M_t of active focal points such that,*

(i) $|M_t| \geq \left\lceil \frac{n-f}{2} \right\rceil - 1$ (Equality holds only if /focal points are faulty and one is recovering);

(ii) The state $\bar{s}$ of its *active* focal points at time *t* is such that $s_1 \leq_s s$.

Proof. Let us denote t_1 — *res* (o_1), and $I = [t_1, t]$. We prove the lemma by induction on the number *k* of subintervals $W_1, \ldots,$ $W_i, \ldots, W_k$ of *I* of size $\leq\tau$, such that $W_i = [t_1 + (i - 1)_\tau, t_1 + i_\tau]$ for $i = 1, \ldots, k$, and $[t_1, t_2]$ C $U^k_{i=1}W_i$ .We want to show that at any time *t* there exists a subset M_t satisfying definitions 1. And 2.
If $k = 1$, there exists a subset M_t of active focal points whose ~~s~~tate is $\geq s_1$. It consists of the reply set *Q* associated with o_1, less an eventual additional failure occurred in $[t_1, t]$.Therefore, because of Lemma 6 and Assumption 1 and 2 of our failure model.

$$|M_t| \geq \left\lceil \frac{n-f}{2} \right\rceil - 1$$

The equality holds only if $f + 1$ focal points in *Q* did not receive $o_1$request and one of the focal points in *Q* fails during $[t_1, t]$. This can occur only if one focal point recovers, because of Assumption 1. In addition, the state of any recovering focal point in W_1 is $\geq s_1$ because $M \cap Q \neq \emptyset$ for each $Q \in Q_m$ In fact

$$|Q \cap M_t| \geq \left\lceil \frac{n-f}{2} \right\rceil + \left\lceil \frac{n+f+3}{2} \right\rceil - n - 1 \geq 1$$

Therefore each focal point that recovered during W_1 can be accounted in set M_t after its recovery. Therefore,

$|M_t| = \left\lceil \frac{n-f}{2} \right\rceil - 1$ Only if *f focal* points are faulty and one is recovering.

5. Problem Specification

The methodology for formal specification of the geoquorum approach is illustrated by considering a set of mobile nodes with identifiers with values from 0 through (N-1) moving through space. Initially some of the nodes are idle while others are active. Nodes communicate with each other while in range. A node can becomes idle at any time but can be reactivated if it encounters an active node. The basic requirement is that of determining that all nodes are idle and storing that information in a Boolean flag (claim) located on some specific node say node (i_0), formally the problem reduces.

Stable W (S$_1$)

Claim detects W (P$_1$)

Where W is the condition

W=< ^ i: 0 < i ≤N:: idle [i] > ^{1} (D$_1$)

(S$_1$) is a safety property stating that once all nodes are idle, no node ever becomes active again. (P$_1$) is a progress property requiring the flag claim to eventually record the system's quiescence. Using idle [i] to express the quiescence of a node and define active [i] to be its negation. It is important to note that the problem definition in this case does not depend on the nature of the underlying computation.

5.1 Formal Derivation

In this section, specification refinement techniques are employed toward the goal of generating a programming solution that accounts for the architectural features of ad hoc networks which form opportunistically as nodes move in and out of range. The refinement process starts by capturing high level behavioral features of the underlying application [14] [15]. In each case, we provide the informal motivation behind that particular step and show the resulting changes to the specification. As an aid to the reader, each refinement concludes with a list of specification statement labels that captures the current state of the refinement process, as in: Refinement 0: P_1, S_1.

5.1.1 Refinement 1: Activation Principle

A node invocation may become put, get, confirm, reconfig – done. The safety property of these computations can be expressed as:

Then Get [i]

(S_2)

Unless

$< \exists id: \ \text{config-id} \neq \text{new-config-id} :: \text{config-id} > \text{new-config-id} >$

Put [i] (S_3)

Unless

$< \exists id, T: \ \text{config-id} \neq \text{new-config-id}, Tag \neq \text{new-tag} :: \text{config-id} >$
$\text{new-config-id}, Tag > \text{new-tag})$

Confirm [i] (S_4)

Unless

$< \exists T: \ Tag \neq \text{new-tag} :: Tag > \text{new-tag} >$

Recon-done[i] (S_5)

Unless

< ∋ id, ip: config-id ≠ new-config- id:: recon-ip=false>

In previous, it is determined that all invocations and its conditions of the application. Refinement 1: P_1, S_2, S_3, S_4, S_5.

5.1.2 Refinement 2: Parameters Based Accounting

Frequent node movement makes direct counting inconvenient, but we can accomplish the same thing by associating id, port-number with each invocation node. Independent of whether it is currently idle or active, each node in the system holds zero or more ids. The advantage of this approach is that ids can be collected and then counted at the collection point. If we define D to be the number of ids in the system and I to be the number of confirm nodes, i.e.

$D \equiv < +i :: id[i] >$ (D_2)

$I \equiv < +i : confirm[i] :: 1 >$ (D_3)

The relationship between the two is established by the invariant:

Inv.D = I (S_6)

By adding this constraint to the specification, the quiescence property (W) may be replaced by the predicate (D = N), where N is the number of nodes in the system. Property (P_1) is then replaced by:

Claim detects D = N (P_2)

With the collection mechanism left undefined for the time being. Refinement 2: P_2, S_2, S_3, S_4, S_5, S_6.

5.1.3 Refinement 3: Config-Ids Increasing

To maintain the invariant that the number of ids in the system reflects the number of idle nodes, activation of an idle node requires that the number of ids increase by one. Therefore, when an active node put invoke an idle node, they must increase config-id between them. To express this, we add a history variable config-id $'$ [i], which maintains the value config-id [i], held before it changed last. And the put invocation is a history state of the put-ack state node; this for all states of nodes of the related work and the safety property of the put invocation is as follow:

Put [i]

(S_7)

Put-ack [j]

Unless

$< \exists j : j \neq i$ put [j] $\wedge$ put - ack [j] $\wedge$

put _ Ack[i] $\wedge$ (new - Config _ id [i] + new _ Config - id [j] =

Config _ id [i] + Config _ id[j] + 1 $\geq 0 >$

Captures the requirement that, when node *i* activates and becomes node *j*, the config-id of node *j* must be increase in the same step. Clearly, this new property strengthens property (S_2), (S_3), (S_6). Refinement 3: P_2, S_4, S_5, S_7.

5.1.4 Refinement: Operations –Id Collection

According to FPE server, FPE client algorithms, the completed operations is arranged in rank of operations that have been simulated initially ø and there exist Val ∈ V, holds current value of the simulated atomic node, initially v_0. oid is a history id of the operation; new-oid is the current id of the operation to simplify our narration, a host with a higher id is said to rank higher than a node with a lower id. Oid (v_0) should eventually collect all N oids. We will accomplish this by forcing nodes to pass their oids to lower ranked nodes for this, we introduce two definitions:

L=<+i: obj [i]:: oid [i] > (D4)

Count the number of oids idle agents hold. Obviously, L = N, when all nodes are idle. We also add

w= < max i: L=N ^ oid [i] > 0:: i > (D5)

To keep track of the highest ranked node holding oids. After all nodes are idle, we will force *w* to decrease until it reaches 0.when *w*=0 and L=N, obj (v_0) will have collected all the oids. At this stage we add a new progress property,

$$w = k > 0 \text{ until } \mathrm{w} < \mathrm{k} \qquad (\mathrm{P}_3)$$

That begins to shape the progress of oid passing. As mentioned, the until property is a combination of a safety property (the unless portion) and a progress property (the leads – to portion). As long as the highest ranked oid holder passes its oids to a lower ranked node, we can show that all the oids will reach obj (V_0= 0) without having to restrict the behavior of any node except node (*w*) = obj (*w*). Some can replace (P_2) with

Claim detects (*w*=0) (P_4)

Refinement 4: P_2, P_3, S_4, S_5, S_7.

5.1.5 Refinement 5: Pairwise Communication

According to the code for the FPE client and FPE server which discussed in section 3.2 clearly, a node can only activate another node or pass (join-ids=jid) to another node if the two parties can communicate. To accomplish this, we introduce the predicate, com (i, j) that holds if and only if nodes *i* and *j* can communicate.

We begin this refinement with a new safety property:

Idle [i]

(S_8)

Unless

< ϶ j: $j \neq$ i : : active$^{/}$[j] ^ active [j] ^ active [i] ^ (join-id[i] + join-id [j]=(join-id)$^{/}$[i] +(join-id)$^{/}$[j]-1)> 0 ^ com (i,j)>

This requires that nodes i and J be in communication when J activates i. Also, adding the property:

Join-id [j] > 0 ^ L=N ^ $j \neq$ 0 (P_5)

Until

Join-id [j] = 0 ^ L= N ^ < ϶ i< j:: com (i, j)>

This requires a node to pass its jids and when it does, to have been able to communicate with a lower ranked node. As we leave this refinement, we separate property (P_5) into its two parts; a progress property,

Join –id [J] > 0 ^ L = N ^ J ≠ 0 (P_6)

Leads-to

Join-id[j] > 0 ^ L = N ^ < ϶ I < j:: com (i, j)> , And a safety property,

Join-id [j] > 0 ^ L= N ^ $j \neq$ 0 (S_9)

Unless

Join-id[j] = 0 ^ L=N ^ < Э i< j:: com (i, j)>

Refinement 5: P_2, P_3, P_6, S_4, S_5, S_7, S_8, S_9

5.1.6 Refinement 6: Contact Guarantee

Property (P6) conveys two different things. First; it ensures that a node with join-ids will meet a lower ranked node, if one exists. Second, it requires the join-ids to be passed then to the lower ranked node.

The former requires us to either place restrictions on the movement of the mobile nodes or make assumptions about the movement. For this reason, we refine property (P_6) into two obligations.

The first: Join-id [J] > 0 ^ L = N (P_7)

Leads – to

Join-id [j] > 0 ^ L = N ^ < Э i< j:: com (i,j)>

Guarantees that a node with join-ids will meet a lower ranked node.

The second, Join-id [j] > 0 ^ L = N ^ < Э i< j:: com (i, j)>

Lead-to

Join-id [j] = 0 ^ L = N ^ < Э i< j:: com (i,j)>

Forces a node that has met a lower ranked node, to pass its join-ids. At the point of passing, communication is still available. There two new properties replace property (P_6).

Refinement 5: P_2, P_3, P_7, P_8, S_4, S_5, S_6, S_7, S_8, S_9

6. Performance Evaluation for Geoquorum Approach: Implementing Atomic Read/ Write Shared memory in Mobile Ad hoc network using fuzzy logic.

Let us consider these assumptions:
1-Input status word descriptions
Almost no- connect
About right
Connect
2- Output action word descriptions
Ack- response
No change needed
Almost no- response
3- Rules
Translate the above into plain English rules (Called linguistic Rules). These rules will appear as follow:
Rule 1: If the status is connect then Ack – response.
Rule 2: If the status is about right, then no change need
Rule 3: If the status is almost no- connect then Almost no- response.
4- The next (3 steps) use a charting technique, one function of the charting technique is to determine "The degree of membership" of: Almost no- connect, about right and connect triangles for a given values (see fig.6).

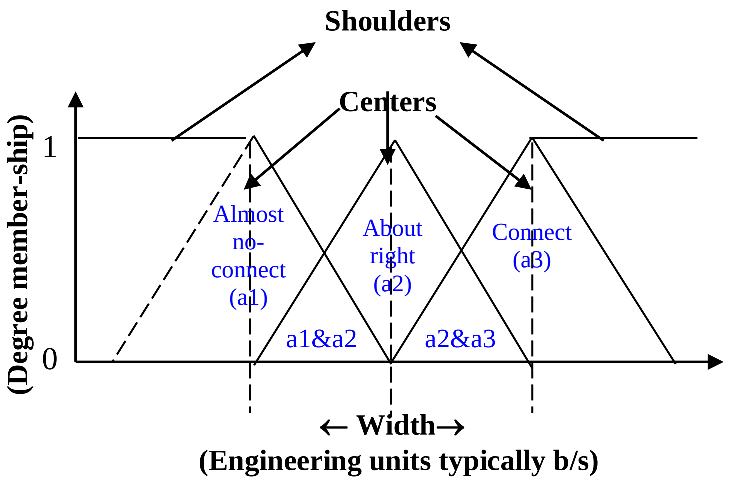

Fig (6) Membership Functions

5- Associate the above inputs and outputs as causes and effects with a rules charts, as in the next fig.7 below, the chart is made with triangles, the use of which will be explained. Triangles work just fine and are easy to work with width of the triangles can vary. Narrow triangles provide tight control when operating conditions are in the area. Wide triangles provide looser control. Narrow triangles are usually used in the center, at the set point (the target value).

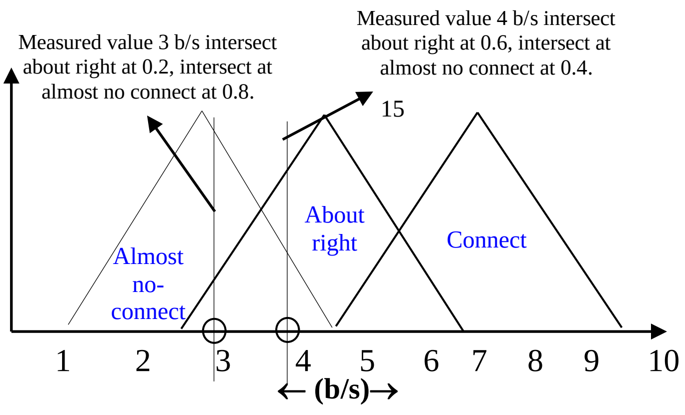
Measured value 3 b/s intersect about right at 0.2, intersect at almost no connect at 0.8.
Measured value 4 b/s intersect about right at 0.6, intersect at almost no connect at 0.4.
15
Almost no-connect
About right
Connect
1 2 3 4 5 6 7 8 9 10
← (b/s)→

Fig (7) Cause - Effect

6- “Effect” (output determining) triangles is drawn with their value (h=3 b/s or 4 b/s and their multiplications) is determined. The triangles are drawn by the previous rules. Since the height doesn’t intersect with connect, these “effect” triangles will be used to determine the controller output.

Conclusions and Future Work

In this paper the formal derivation of geoquorum approach is represented for a mobile ad hoc network. This formal specification is built from mobile unity primitives and proof logic .Also, this paper provides strong evidence that a formal treatment of mobility and its applications isn't only feasible but, given the complexities of mobile computing. There is an open area for using this specification to mechanistically construct the program text as: first, defining the program components, and then deriving the program statements directly from the final specification, the resulting program (called a system in mobile UNITY). Also, we have viewed a small set of mobility constraints that are necessary to ensure strong data guarantees in highly mobile networks. Quorum systems in highly mobile networks are discussed and devised a condition that is necessary for a quorum system to guarantee data consistency and availability under our mobility constraints as a survey. This work leaves several open questions such as the problem of dealing with network partitions and periods of network instability in which the set of assumptions are invalid.

REFERENCES

[1] A. Smith, H. Balakrishnan, M. Goraczko, N. Priyantha, "Support for Location: Tracking Moving Devices with the Cricket Location System", in: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services, Jun 2014.

[2]S. Gilbert, N. Lynch, A. Shvartsman, "RAMBO II: Rapidly Reconfigurable Atomic Memory for Dynamic Networks," in: Proceedings of the International Conference on Dependable Systems and Networks, June 2013, PP. 259-269.

[3]B. Liu, P. Brass, O. Dousse, P. Nain, D. Towsley, "Mobility Improves Coverage of Sensor Networks", in: Proceedings of Mobile Ad Hoc, May 2015, PP. 300-308.

[4]R. Friedman, M. Gradinariu, G. Simon, "Locating Cache Proxies in MANETs", in: Proceedings of the 5th International Symposium of Mobile Ad Hoc Networks, 2014, PP. 175-186.

[5]J. Luo, J-P. Hubaux, P. Eugster," Resource Management: PAN: Providing Reliable Storage in Mobile Ad Hoc Networks with Probabilistic Quorum Systems", in: Proceedings of the 4th International Symposium on Mobile Ad Hoc Networking and Computing, 2019, PP. 1-12.

[6]D. Tulone, Mechanisms for Energy Conservation in Wireless Sensor Networks. Ph.D. Thesis, Department of Computer Science, University of Pisa, Dec 2015.

[7]W. Zhao, M. Ammar, E. Zegura,"A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks", in: Proceedings of the 5th International Symposium on Mobile Ad hoc Networking and Computing, May 2014, PP.187-198.

[8]S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, J. Welch, "GeoQuorums: Implementing Atomic Memory in Mobile Ad Hoc Networks", in: Proceedings of the 17th International Conference on Distributed Computing, October 2018, PP. 306-320.

[9]H. Wu, R. Fujimoto, R. Guensler, M. Hunter, "MDDV: A Mobility-Centric Data Dissemination Algorithm for Vehicular Networks", in: Proceedings of the1st International Workshop on Vehicular Ad hoc Networks, Oct 2017, PP. 47-56.

[10]J. Polastre, J. Hill, D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks", in: Proceedings of IJCS PP: 231-245, 2014.

[11]S. PalChanduri, J.-Y. Le Boudec, M. Vojnovic, "Perfect Simulations for Random Mobility Models", in: Annual Simulation Symposium 2015, PP. 72-79. Available from: http://www.cs.rice.edu/santa/ research/mobility

[12]J.Y. Le Boudec, M. Vojnovic, "Perfect Simulation and Stationarity of A Class Of Mobility Models", IJCSS, pp:225-235 (2016).

[13]T. Hara, "Location Management of Replication Considering Data Update in Ad Hoc Networks, in: 20th International Conference AINA, 2019, PP. 753-758.

[14]T. Hara," Replication Management for Data Sharing In Mobile Ad Hoc Networks", Journal of Interconnection Networks 7(1) (2016), PP.75-90.

[15] Y Sawai, M. Shinohara, A. Kanzaki, T. Hara, S. Nishio, "Consistency Management Among Replicas Using A Quorum System in Ad Hoc Networks", MDM (2018), PP.128-138