

# Relying on Safe Distance to Achieve Strong Partitionable Group Membership in Ad Hoc Networks

Qingfeng Huang, Christine Julien, Gruia-Catalin Roman

Qingfeng Huang, Christine Julien and Gruia-Catalin Roman are with Department of Computer Science and Engineering, Washington University, St. Louis, MO 63130, USA. Emails: {qingfeng,julien,roman}@cse.wustl.edu

June 13, 2003

DRAFT

## Abstract

The design of ad hoc mobile applications often requires the availability of a consistent view of the application state among the participating hosts. Such views are important because they simplify both the programming and verification tasks. We argue that preventing the occurrence of unannounced disconnection is essential to constructing and maintaining a consistent view in the ad hoc mobile environment. In this light, we provide the specification for a partitionable group membership service supporting ad hoc mobile applications and propose a protocol for implementing the service. A unique property of this partitionable group membership is that messages sent between group members are guaranteed to be delivered successfully, given appropriate system assumptions. This property is preserved over time despite movement and frequent disconnections. The protocol splits and merges groups and maintains a logical connectivity graph based on a notion of safe-distance. An implementation of the protocol in Java is available for testing. This work is used in an implementation of LIME, a middleware for mobility that supports transparent sharing of data in both wired and ad hoc wireless environments.

## Keywords

Mobility, ad hoc network, group membership, consistency.

## I. INTRODUCTION

Group membership has been an important problem in the area of fault-tolerant distributed computing and has been the subject of extensive investigation [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. Solving the problem requires the provision of a service that establishes and maintains some kind of agreement over time among participating components about who is currently in the group, despite the presence of failures in the corresponding distributed system. Such a group membership service simplifies the development of many fault-tolerant distributed applications [1] and is widely used for supporting reliable group communications.

We encountered a group membership problem in our attempt to support group computation in ad hoc mobile environments. Peer-to-peer or group cooperation are common scenarios for ad hoc mobile applications. When two or more mobile hosts come together to form a group working on the same problem, it is sometimes essential for all of them to have the same view of the joint computation state when they start working or when some of the members leave the group. One important piece of group state information is membership in the group, i.e., who is and who is not in the working group. Any sit-

uation that demands (for legal or technical reasons) the presence of two or more specific entities to carry out a task may impose the need for a consistent membership view. One can envision the futuristic notion of an electronic witness to a contractual transaction or the circumstance in which routine maintenance of commercial aircraft requires secure logging in the presence of an FAA inspector carrying an authorized electronic badge. The need for consistent view among group member also rises in the fledging area of intelligent transportation systems [12], [13]. Driving assistance systems are being developed on top of the mobile wireless ad hoc network for collision warning and avoidance [14], [15], [16]. In such systems, a consistent view about participating vehicles and consistent decisions across the members about acceleration, deceleration, lane change [17], etc., are critical. The importance of agreeing on the same view about membership state introduces the need for group membership services, which maintain a consistent view about the membership among group members.

The group membership problem we encountered in the mobile ad hoc environment is new in the sense that it has special requirements that are different from all previously studied ones. It not only requires availability and progress in the presence of network partitions, as most partitionable group membership services do, but also requires consistency when partitions occur, which none of the previous partitionable group membership services support [18], [10], [19], [5], [20], [21], [8]. The reason why the previous partitionable group membership services do not support consistency in the presence of partitions is fundamental; because the assumed system model is asynchronous and distributed, consensus is impossible [22], [23]. Furthermore, network partition in a fixed network is usually infrequent and short-lived. This makes manual checking a viable option to solve any inconsistencies that might occur during a network partition. Yet in our case, we emphasize the consistency requirement because network partition is a frequent event in ad hoc mobile environments and the cost of “short-term” inconsistency can be very high in mobile computation scenarios. Mobile hosts interact over wide spaces, and inconsistency can propagate indefinitely and cause irreparable damage in mission critical applications. Similar strict consistency has been considered by Cristian and Fetzer [24] for the primary group membership problem in timed-asynchronous systems, but has not been investigated

for the partitionable group membership problem.

The goal of our group membership service is not only to create a consistent view of group membership among participants, but also to help users and application programmers avoid the complexities introduced by potential data inconsistencies caused by mobility-induced link failures. Mobility-induced link failures refer to the communication failures caused by mobile units moving out of each other's communication range. The key characteristic of a mobility-induced link failure is that it is unrecoverable and more damaging than link failures in fixed networks. For instance, a message sent while a physical network partition is taking place can be in a dubious state of delivery. That is, one side (sender/receiver) thinks it is "delivered," and the other side (receiver/sender) thinks it is not. To make matters worse, no mechanism exists for either party to disambiguate this situation. The link failures occurring in fixed networks are usually recoverable, in the sense that the communicating peers can always use an acknowledgement-retransmission-based protocol like TCP to ensure a link failure is transient, because in a fixed network any failed link will "eventually come back." The unrecoverability of mobility-induced link failure can result in permanent data inconsistency and poses a great challenge to mobile application programmers. Our group membership service tries to help programmers in this matter by guaranteeing that communication between group members will not suffer from mobility-induced failure.

The strong requirements of the new group membership service make it impossible to implement in asynchronous systems. To make this strong group membership problem solvable, we introduce a certain level of synchrony into our system model. We assume the communication service is reliable in each physical network partition and has a bounded message delivery time  $t_d$  within the partition. Furthermore, as we mentioned earlier, a message sent at time  $t < t_d$  before a physical network partition takes place can be in a dubious state of delivery. We introduce a key concept called *safe distance* to solve this problem. It works by preventing a group message from falling into a pocket of network instability caused by partitioning. We rely on location information to decide when a host within communication range is admitted to or eliminated from a group. The group membership policy is conservative in nature to ensure that the changes in group membership

appear to be atomic, i.e., are serializable transactions. The algorithm accommodates both the merging of groups and the partitioning of one group into multiple disjoint groups.

We have implemented a version of the strong partitionable group membership protocol in Java. It supports LIME [25], [26], a middleware for rapid development of mobile applications. The implementation works properly if the system assumptions of the protocol are met. The validity of the system assumptions and how they might be implemented are discussed later in the paper.

In the next section, we formally specify the system requirements and the definition of the group membership service. In section 3, we introduce the concept of safe distance and present our solution strategy for the group membership problem. Section 4 describes an implementation of our protocol. Section 5 analyzes the relationships between safe distance, network delay, and mobile host speed. Discussions and conclusions appear in Sections 6 and 7, respectively.

## II. PROBLEM DEFINITION

Our ultimate goal is to provide application developers with the ability to maintain a consistent global data structure in a setting in which mobile hosts come and go as they please and engage in reliable transient collaborative activities. In this context, the group membership service needs to provide an accurate snapshot of the membership view all the time, and a message entrusted in a view shall be guaranteed to be delivered to members in that view, despite motion and motion-induced disconnections. This property makes the group membership service useful for many mobile applications, such as those we mentioned earlier. Next we seek to formally define the group membership problem.

### A. Membership Specification

The group membership service can be specified by defining its local state variables and the safety and progress properties that it satisfies. We use terminology and notation similar to that of Cristian [27] to specify the properties. Let  $P$  be the set of all hosts that exist over time. We assume each host has a unique identifier denoted by  $i$  and drawn from the integer set  $Z^+$ , and all groups that exist over time have identifiers drawn from a partially ordered set  $G$ . Each host in  $P$  maintains the following two state variables:  $g$

and  $\pi$ .  $g$  is the group identifier, and  $\pi$  is a subset of  $P$ .  $\pi$  is also called the membership view of  $P$ , or “view” for short. Let  $T = [0, \infty)$  be the range of time. Two functions are introduced to simplify the phrasing of the specification:

$$group : P \times T \longrightarrow G$$

$$mem : P \times T \longrightarrow 2^P$$

$group(p, t)$  yields the group identifier for host  $p$  at local time  $t$ ;  $mem(p, t)$  yields  $p$ 's local view of the membership  $\pi$  at time  $t$ . We call a group  $g$  if its identifier is  $g$ . We call  $g'$  a successor of group  $g$  if there exists a member  $p$  of  $g$  such that the next group  $p$  joins after leaving  $g$  is  $g'$ . Like in [27],  $succ(g, p)$  is used to denote the successor of group  $g$  relative to host  $p$ ;  $pred(g, p)$  is used to denote the predecessor of group  $g$  relative to  $p$ . Given these terms, the group membership service is specified in the following manner:

- *Self inclusion*: a host is always a part of its membership view, i.e.,  $p \in mem(p, t)$ <sup>1</sup>
- *Local monotonicity*: group identifiers installed on each host are in increasing order, i.e.,  $pred(g, p) < g < succ(g, p)$ .
- *Initial membership view*: a host always installs itself as the only member in its view when it starts, i.e.,  $mem(p, t_{init}) = \{p\}$ .
- *Membership Agreement*: If hosts  $p$  and  $q$  have the same group id, then they have the same views, i.e.,  $group(p, t_p) = group(q, t_q) \Rightarrow mem(p, t_p) = mem(q, t_q)$ .
- *Membership change justification*: The successor of group  $g$  with respect to  $p$  is either a proper superset or a proper subset of the group  $g$ .
- *Same view message delivery*: If host  $p$  sends a message  $m_{pq}$  to host  $q$  at time  $t$ , and  $q$  is in  $mem(p, t)$ , then  $m_{pq}$  is guaranteed to be delivered to  $q$  at time  $t'$ , and  $mem(q, t') = mem(p, t)$ . Note that this property is traditionally included in group communication service ([18][10]) rather than the group membership service([1][10]). We will explain later why we include it here.
- *Conditional bounded integration*: Let  $p$  belong to group  $g_1$  and  $q$  belongs to group  $g_2$ . Assume  $g_1$  and  $g_2$  are the only two groups satisfying the merging criteria (explained later) at time  $T$ . There exists a time constant  $T_c$  such that, if  $g_1$  and  $g_2$  stay satisfying the

<sup>1</sup>To simplify notation, we assume unrestricted variables are universally quantified

merging criteria for at least time  $T_c$ , the two groups will merge into one group, i.e.,  $\exists t_p, t_q \in [T, T + T_c], mem(q, t_q) = mem(p, t_p)$ .

- *Conditional group split:* A group splits only if it is necessary, i.e., when a split condition (explained later) is met.

The first two safety requirements are common to most partitionable group membership specifications [20], [6], [18], [10], [11]. The third requirement in our specification, the initial membership view requirement, differs from most of those in the literature [20], [6], [18], [10], [11] and is relatively unique, catering to the reality of ad hoc mobile environments: a mobile host may start up with no knowledge of other hosts in the world. The membership change justification is introduced to ensure some continuity in view change properties.

The same view message delivery requirement is introduced to add more predictability to the group membership service for some applications (e.g. the LIME [28], [25], [26] middleware for mobility). With this unique property, application developers using the service are assured that message delivery is reliable within the scope of the view. In other words, within the membership view, programmers no longer need to worry about the complexity of potential inconsistencies caused by message loss due to mobility. Note that this property (or a similar one) is traditionally included in group communication service ([18],[10]) rather than the group membership service([1],[10]). The reason that we include it here is to augment the group membership service with a property needed by target applications that do not require a full blown group communication service.

The conditional eventual integration and the conditional group split are introduced to avoid the classical problem of “capricious split” [19]. Without requiring bounded integration, a group membership implementation can simply not perform any merging of groups and still satisfy the specification by keeping all the groups singletons. Without the conditional split requirement, a group membership protocol might continuously merge and split groups without justification, yet still satisfy the specification. Note that in the specification we did not explicitly say what the merging criteria and the split criteria are. The merging and split criteria in general are application dependent. In this paper, we use the weakest merging and split criteria. The merging criterion is weakest in the sense that it only requires the group membership properties to be satisfied for the new group. The split

criteria is weakest in the sense that the group splits only if the group membership property cannot be guaranteed without doing so. No other condition outside of the membership specification is forced to hold.

### B. System Model

Our system model assumes that there are no host crash failures and no omission and performance failures caused by network congestion. The only failure in our system model is the communication failure caused by hosts moving out of each other's communication range. This model is a reasonable starting model for ad hoc mobile systems for two reasons. First, mobility-induced disconnection is much more frequent than host crash failure given current hardware and software technology. Second, a mobile network in theory can be equipped with enough bandwidth for communications needed by the applications on top of it, such that congestion can be made a rare event compared to the occurrence of partitions.

Our system model also assumes that the underlying communication service is reliable and timely [29], in the sense that a message entrusted to it is guaranteed to be delivered within a time bound  $t_d$  if the sender and the recipient are physically connected during that time. By physically connected we mean two hosts are either within each other's communication range or transitively connected through other hosts.

For simplicity, we assume that all hosts have the same communication radius and the communication links are bi-directional. We also assume that all hosts know their physical location all the time. We assume no knowledge of the mobility patterns of the mobile hosts except that the movement is continuous in space and has some upper bound  $V_{max}$  on speed. We purposely choose this extreme case in order to explore the limits imposed on the membership problem by random ad hoc mobility. The basic ideas behind our solution strategy are explained in the next section.

## III. SOLUTION STRATEGY

Key to our strategy to implementing the strong group membership is the notion of *safe distance* among hosts and groups, i.e., the idea that if hosts are "close enough", disconnection is not possible for some time and that if they are "just far enough" there is plenty of time to carry out a configuration change before disconnection occurs. In other words,

we define a logical connectivity graph over the physical connectivity such that an edge appears in the logical graph if and only if two hosts represented by corresponding vertices are within safe distance. Group membership reflects partitions in the logical connectivity graph rather than partitions in the physical connectivity graph. In the remainder of this section we explain the safe distance concept and present the discovery and reconfiguration protocols.

#### A. Key Concept: Safe Distance

Given two mobile hosts equipped with compatible wireless transmitters of equal range  $R$ , we state that the distance between them is a safe distance if it does not exceed a threshold  $r(v, t, t')$ , defined as the maximum distance at which one can guarantee that any communication task that takes at most  $t$  time units can be completed with certainty, assuming the two hosts move randomly at a speed that does not exceed  $v$ , and the upper bound for a single atomic configuration change is  $t'$ . Clearly, safe distance cannot be defined in absolute terms but must be considered relative to a context having certain mobility and application characteristics. For example, in Figure 1(a) mobile hosts  $a$  and  $b$  are within communication range ( $R$ ), i.e., they are able to talk to each other directly. They

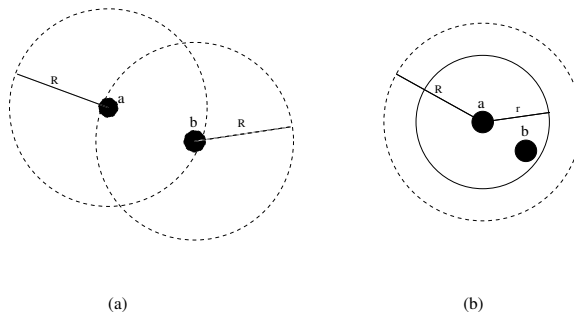


Fig. 1. An example of safe distance

may want to be in the same group and start coordination or resource sharing immediately. Yet, they cannot do so at this point if they wish to guarantee message delivery within the group. This is because  $a$  and  $b$  can move out of each other's range immediately after acknowledging membership in the same group, with the result being that messages between them could not be delivered successfully. The problem arises from the mobile nature of the hosts and the asynchronous nature of message passing. Our solution is to require  $a$

and  $b$  to agree on membership in the same group only when they are “close enough” (as in Figure 1(b)), i.e., they are at a distance

$$r \leq R - 2v * (t + t') \quad (1)$$

In this context,  $t$  is the upper bound for network latency (because the consistency requirement is reliable message delivery) and  $t'$  is the time needed for a group level operation (merge or split) already in progress to complete. The factor  $2v$  accounts for the worst case movement pattern, i.e., a situation in which  $a$  and  $b$  are moving in opposite directions at maximum speed. One can readily see that with this restriction, the reliable message delivery between group members is guaranteed because it takes more than  $t + t'$  time for the two group members to become physically disconnected, no matter how they move. Within this time any message delivery completes even if a configuration change is in progress.

We call a group safe if any two members of the group are connected via a path along which all consecutive hosts are at a safe distance. We extend the notion of safe distance from pairs of hosts to pairs of (safe) groups by requiring that at least two hosts, one in each of the two groups, are at a safe distance. While this definition seems to assume that the safe distance is independent of group size, this assumption is generally not true because both simple message delivery and reconfiguration actually depend on the number of hops that messages must traverse en route. Because the time bound on message-passing depends on the group size, our approach works only when the group size is limited by the nature of the application or is constrained by the reconfiguration protocol.

The concept of safe distance is used to determine when two groups can be merged and when a group must be split in order to maintain the requirements for group membership. To determine whether two groups are within safe distance, one must know the location of all hosts in the region. Since it is too expensive for everyone to keep track of the location of others all the time, we designate a leader for each group. All hosts in a group constantly report their location to the leader, and the leader keeps the map ( $\Pi$ ) of the group, checking constantly to see if the group members are still within safe distance of each other and whether new hosts are present in the region. The map of a group records the spatial location of group members.

## B. Group Discovery Protocol

As defined in the membership specification, each mobile host is given a unique host identifier  $id$ , and starts as a singleton group containing itself as the only member.

For a group to merge with another group, it must first be able to discover which other groups are present in its vicinity. The discovery protocol carries out this function and serves as a supporting layer for the group membership maintenance protocol, i.e., the reconfiguration protocol. In our discovery protocol, hosts in each group use safe distance as a criteria for finding out who is close enough to be a merge candidate, and they report any positive discoveries to their leader. For simplicity, the host with the smallest identifier in a group is chosen as the group's leader. For convenience, we also choose the host identifier ( $id$ ) of the leader to serve as the name for its group ( $gid$ ). Note that  $gid$  is not the same as the partially ordered group identifier  $g$  used in the membership specification. Rather,  $gid$  combined with a group change sequence number  $\tau$  (discussed in more detail later in the paper) yields the partially ordered group identifier  $g$ .

Our discovery mechanism requires every host to periodically broadcast a hello message which contains its location information ( $xy$ ) and its group identifier ( $gid$ ). When two groups move close, several members of one group may receive hello messages from members of the other group. When a host  $u$  receives a hello message, it checks the sender's group identifier and location. If  $u$  finds the sender, say  $v$ , to be a member of another group located within safe distance,  $u$  passes on the information to its group leader that, in turn, will use it for merge related operations. As all group members are involved in discovery, it is possible for the group leader to receive multiple copies of the same notification regarding the appearance of one host. Duplicates are discarded.

There are several things one can do to reduce discovery costs. First, each host may attach discovery information to its periodic location updates to the leader rather than send them separately. This pushes the discovery information towards the leader almost for free, since the location and new neighbor information represent only a few bytes that fit easily in a single packet. The cost associated with this piggy-backing method is the need for each host to keep a short-term memory ( $\xi$ ) of newly discovered neighbors. Second, by utilizing neighborhood information already available at the MAC layer, a host may send

neighbor greetings only when the MAC layer discovers a new neighbor. This reduces the discovery cost significantly in the case when the network topology changes infrequently. The drawback for this method is its dependence on the implementation of the MAC layer on the specific host supporting the application. We chose not to do so in our prototype.

The group discovery protocol allows the group leader to maintain a list of groups which are close enough to be considered for merging. We present the group merging protocol in the next section.

### *C. Reconfiguration Protocol*

The reconfiguration protocol is the key layer in our group membership service. It seeks to merge groups in contact and to split groups that can no longer stay together. From the information collected in group discovery, a leader may find that there are one or more potential candidate groups in its vicinity suitable for merging. If so, it will initiate merging negotiations with the set ( $\Theta$ ) of candidates. Once an agreement is reached regarding who is to participate and who is responsible for coordinating the merge, all affected hosts receive a formal notification about the configuration change from the coordinator. Furthermore, after a host receives a group change notification, in order to prevent messages sent in one configuration from being processed in a different configuration, it must perform a barrier synchronization. One way of accomplishing this is to flush the messages in transit before doing anything in the new configuration. In addition, the participants need to delay the processing of messages arriving from “future” configurations until the synchronization is completed. Message delaying can be accomplished by tagging each message with a configuration sequence number ( $\tau$ ). Flushing requires the participating hosts to send extra messages whose arrivals guarantee that no more messages originating from a prior configuration are in transit. The result is an atomic configuration change. Another way of creating the synchronization boundary is by using a time-out delay. Partitioning works in the same way but without any negotiation because it involves only one group at a time. Next, we use several simple examples to illustrate the merging and partition processes.

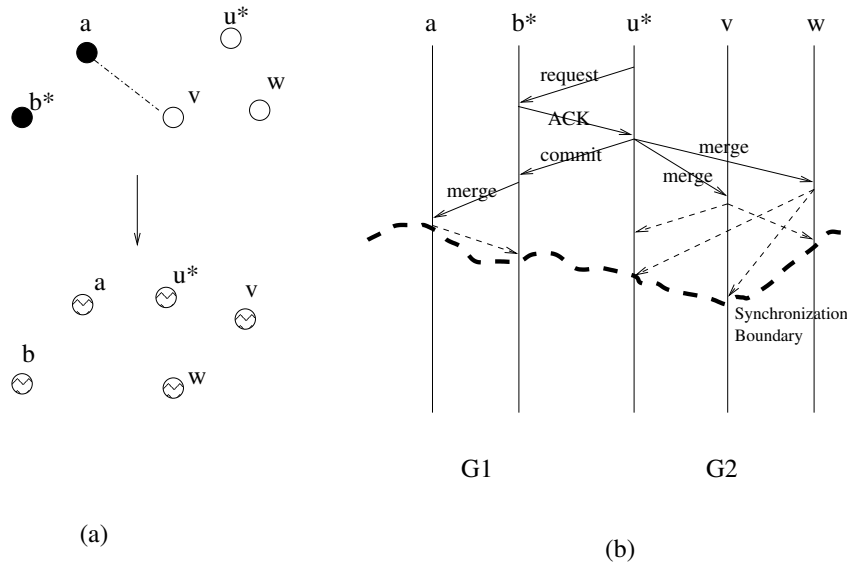


Fig. 2. Merging Process

### C.1 An example of merging

Figure 2 depicts the merging process between two groups, G1 and G2. G1 contains hosts  $a$  and  $b$ , and  $b$  is the leader. G2 contains hosts  $u, v$ , and  $w$ , and has  $u$  as its leader. Assume  $u$  finds G1 to be in its vicinity through the discovery data sent in by  $v$ , and G1 is safe for merging. Next,  $u$  initiates the merger by sending a *merge-request* message to  $b$ , the leader of G1. If willing to participate in the merger,  $b$  sends back an acknowledgment (ACK) along with its group membership list and its configuration sequence number; otherwise, it sends back a disagreement message (NACK), which forces  $u$  to abort the merger with G1. If  $u$  gets back an ACK, as in Figure 2, it generates a new configuration number by adding one to the larger of the current configuration numbers of the two groups. Next, it sends a *merge-commit* to  $b$  and a *merge-order* to its own members. Both the *merge-commit* and *merge-order* messages contain the new group membership list, the new configuration number, and the new leader identity. Upon receiving the *merge-commit* message,  $b$  sends a *merge-order* to its own members. A host enters the flushing phase after it gets a *merge-order* message. It sends a *flush-message* to all other members of its original group and stops sending any other messages until it has received all the expected flush-messages from its group members in the old configuration.<sup>2</sup> After receiving all the expected flush-

<sup>2</sup>The dashed arrows in Figures 2, 3 and 4 represent flush messages.

messages, a host enters the new configuration and all new messages it sends will have the new configuration number in their headers. Clearly, hosts may enter the new configuration at different times. It is possible for a host that is still in the old configuration to receive a message from a host that has already reached the new configuration, as shown in Figure 3. In such cases, the recipient must postpone the processing of this “future” message until

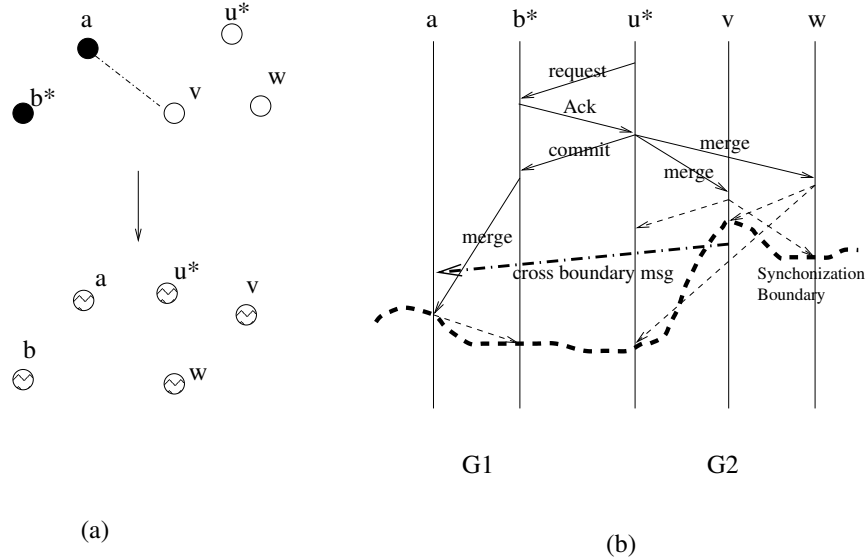


Fig. 3. Synchronization and the Configuration Number

the new configuration is established, thus pretending that the message is “received” in the new configuration. Otherwise, the consistency requirement that messages must be sent and received in the same configuration would be violated. Implementation of this requires a host to check each received message for the configuration in which it was sent before it is processed.

It is possible for  $u$  and  $b$  to initiate the merging at the same time. In this case, a tie-breaking mechanism decides who is to coordinate the merger. We use the *id* as the tie-breaker. The host with the lower *id* aborts its merger request when the collision happens. Additional complications may appear when more than two groups are involved. For example,  $u$  might have entered a merging process with other groups when it receives  $b$ 's *merge-request* message, or it may no longer be a leader because of a merge with other groups or due to a partition process. In all such cases,  $u$  replies with a NACK.

## C.2 An example of partitioning

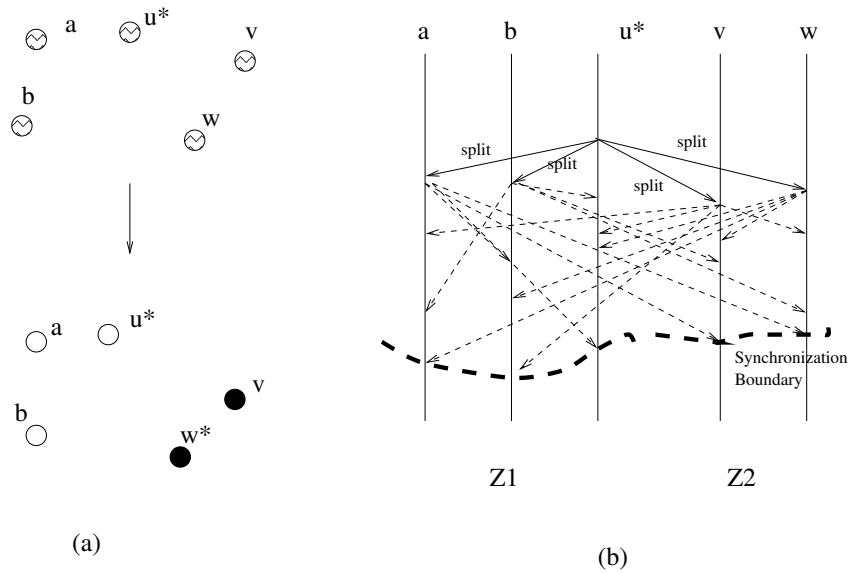


Fig. 4. The Partition Process

Figure 4 illustrates the partition process. Assume that two subgroups of the group  $G$ ,  $Z1$  and  $Z2$ , are moving away from each other. By constantly checking the locations of its group members, the leader  $u$  is able to identify if its group is in a safe spatial configuration, given predefined distance-based safety criteria. Once the leader  $u$  deems the configuration to be no longer safe, it immediately issues a *split-order* message to all the group members. A *split-order* contains three pieces of information: (1) the new leader (*gid*) for the recipient, (2) the new group membership list for the recipient, and (3) the new configuration number, which is the old configuration number incremented by one.  $u$  always chooses the host with the lowest *id* in each subgroup to be the new leader for that subgroup. Upon receiving a *split-order* message, a host enters a message flushing phase, similar to the third phase in the merge process. Each host waits until it is sure that all messages sent to it by group members in the previous configuration are received, either by receiving all the expected flush messages or by employing a time-out delay. Each newly assigned leader assumes its leadership role after the synchronization.

The group leader must check its group configuration frequently enough in order to discover any unsafe situation in a timely fashion. As we will see in the analysis section,

the threshold for safe distance does depend on the checking frequency, in addition to factors discussed earlier. The example above shows a process in which a group partitions itself into two other groups. In general, a leader might find it necessary to split its group into more than two subgroups in order to preserve the safe distance property. The partition process is the same. Next we explain how the leader determines when the group configuration is not safe and how to split it into safe sub groups.

### C.3 Determine safe groups

To determine if its group configuration is safe, the leader maintains a logical connectivity graph. In the logical connectivity graph, two nodes have an edge of weight one between them if the physical distance between them is less than a partition safe distance ( $d_p$ ) and no edge between them otherwise. Whenever a new location is reported, the graph is updated by recomputing all the edges to the reporting node. This takes  $O(N)$  steps per update, where  $N$  is the number of nodes in the group. Given the logical connectivity graph, the depth first search (DFS) algorithm can be used by the leader to determine connected clusters in  $O(N)$  steps. So the total time complexity for maintaining the logical

<u>State Variables</u>	
$id$	node identifier
$gid$	group identifier
$xy$	node location constantly updated by some external mechanism
$\tau$	group transaction sequence number
$\pi$	group member list
$\xi$	the set of newly discovered leaders
$\Pi$	group map containing all members' locations (empty except for the leader)
$\Theta$	the set of merging contacts, all of which are leaders of other groups (empty except for leaders)
<i>UpdateTimer</i>	timer for periodic location update
<i>GreetingTimer</i>	timer for periodic neighbor discovery
<u>Support Functions</u>	
$update(\xi, gid);$	Add $gid$ to the list of newly discovered leaders.
$update(\Theta, \xi');$	Update merge contact list $\Theta$ with newly discovered leaders from a member's report.
$update(\Pi, v, xy');$	Update group map $\Pi$ with group member $v$ 's new location, $xy'$ .
$MergeSafe(\Pi, \Pi', P);$	Verify that the merger of $\Pi$ with $\Pi'$ is safe, according to policy $P$ . $P$ includes safe distance information and the merging status of this group member. For example, if a host is in the process of merging, it is not safe to start a merge.
$ClearOldChannels();$	Clear all group communication channels.
$GeneratePartitions(\Pi, P);$	Generate partitions for $\Pi$ , subject to policy $P$ . This function generates a set of triples of the form $\langle \Pi_{new}, \pi_{new}, gid_{new} \rangle$ .

Fig. 5. State Variables and Support Functions

Actions triggered by changes in the local state	Actions triggered by the arrival of a message
<p>NEIGHBORGREETINGS(<math>u</math>)</p> <p>Precondition:  <i>GreetingTimer</i> expires;</p> <p>Effect:  reset <i>GreetingTimer</i>;  LOCALBROADCAST NEIGHBORHELLO(<math>u, gid</math>);</p>	<p>GET NEIGHBORHELLO(<math>v, gid</math>)</p> <p>Precondition:  true;</p> <p>Effect:  update(<math>\xi, gid</math>);</p>
<p>LOCATIONUPDATE(<math>u</math>)</p> <p>Precondition:  <i>UpdateTimer</i> expires or <math>\xi</math> changes;</p> <p>Effect:  reset <i>UpdateTimer</i>;  SEND INFORMLEADER(<math>u, xy, \xi</math>) to <math>gid</math>;</p>	<p>GET INFORMLEADER(<math>v, xy', \xi'</math>)</p> <p>Precondition:  <math>u</math> is the leader;</p> <p>Effect:  update(<math>\Theta, \xi'</math>);  update(<math>\Pi, v, xy'</math>);</p>
<p>MERGE(<math>u</math>)</p> <p>Precondition:  <math>u</math> is the leader;  <math>\Theta</math> is not empty;</p> <p>Effect:  add <math>u</math>'s members to a new map (<math>\Pi_{new} = \Pi</math>);  add <math>u</math>'s members to a new list (<math>\pi_{new} = \pi</math>);  <b>for</b> each <math>v</math> in <math>\Theta</math>  SEND MERGE-REQUEST(<math>u, \Pi</math>) to <math>v</math>  <b>endfor</b>  <b>for</b> each <math>v</math> in <math>\Theta</math>  <b>if</b> GET MERGINGACK(<math>v, \tau_v, \Pi_v</math>)  update group map (<math>\Pi_{new} = \Pi_{new} + \Pi_v</math>);  update group member list (<math>\pi_{new} = \pi_{new} \cup \{\Pi_v\}</math>);  store <math>\tau_v</math>;  <b>else if</b> GET MERGINGNACK(<math>v</math>)  <math>v</math> will not participate in merger;  remove <math>v</math> from <math>\Theta</math>;  <b>endif</b>  <b>endfor</b>  <b>if</b> <math>\Theta</math> is not empty  set <math>\tau</math> to the max of all <math>\tau_v</math> received;  <b>for</b> each <math>v</math> in <math>\Theta</math>  SEND MERGE-COMMIT(<math>\pi_{new}, gid, \tau_v, \tau_{new}</math>) to <math>v</math>;  <b>endfor</b>  <b>for</b> each <math>w</math> in <math>\pi</math>  SEND MERGE-ORDER(<math>\pi_{new}, gid, \tau_{new}</math>) to <math>w</math>;  <b>endfor</b>  <b>endif</b>  empty <math>\Theta</math>;  update group member list (<math>\pi = \pi_{new}</math>);  update group map (<math>\Pi = \Pi_{new}</math>);</p>	<p>GET MERGE-REQUEST(<math>v, \Pi'</math>)</p> <p>Precondition:  true;</p> <p>Effect:  <b>if</b> <i>MergeSafe</i>(<math>\Pi, \Pi', P</math>)  SEND MERGINGACK(<math>u, \tau, \Pi</math>) to <math>v</math>  empty <math>\Theta</math>;  update safety condition <math>P</math>;  <b>else</b>  SEND MERGINGNACK(<math>u</math>) to <math>v</math>;  <b>endif</b></p>
<p>PARTITION(<math>u</math>)</p> <p>Precondition:  <math>u</math> is the leader;  partition predicted based on location updates;</p> <p>Effect:  <math>\Psi = \text{GeneratePartitions}(\Pi, P)</math>;  <b>for</b> each <math>\langle \Pi_{new}, \pi_{new}, gid_{new} \rangle</math> in <math>\Psi</math>  <b>for</b> each <math>w</math> in <math>\pi_{new}</math>  SEND SPLIT-ORDER(<math>\Pi_{new}, \pi_{new}, gid_{new}, \tau</math>) to <math>w</math>;  <b>endfor</b>  <b>endfor</b>  empty <math>\Psi</math>;</p>	<p>GET MERGE-COMMIT(<math>\pi_{new}, gid_{new}, \tau_u, \tau_{new}</math>)</p> <p>Precondition:  <math>u</math> is the leader;  transaction numbers match (<math>\tau == \tau_u</math>);</p> <p>Effect:  <i>ClearOldChannels</i>();  <b>for</b> each <math>w</math> in <math>\pi</math>  MERGE-ORDER(<math>\pi_{new}, gid_{new}, \tau_{new}</math>) to <math>w</math>;  <b>endfor</b>  update group id (<math>gid = gid'</math>);  update transaction sequence (<math>\tau = \tau_{new}</math>);  update group member list (<math>\pi = \pi_{new}</math>);  empty <math>\Pi</math>;</p>
	<p>GET MERGE-ORDER(<math>\pi_{new}, gid_{new}, \tau_{new}</math>)</p> <p>Precondition:  true</p> <p>Effect:  <i>ClearOldChannels</i>();  update group id (<math>gid = gid_{new}</math>);  update transaction sequence (<math>\tau = \tau_{new}</math>);  update group member list (<math>\pi = \pi_{new}</math>);</p>
	<p>GET SPLIT-ORDER(<math>\Pi_{new}, \pi_{new}, gid_{new}, \tau_{new}</math>)</p> <p>Precondition:  true;</p> <p>Effect:  <i>ClearOldChannels</i>();  update group id (<math>gid = gid_{new}</math>);  update transaction sequence (<math>\tau = \tau_{new}</math>);  update group list (<math>\pi = \pi_{new}</math>);  <b>if</b> <math>u == gid</math>  update group map (<math>\Pi = \Pi_{new}</math>);  <b>endif</b></p>

Fig. 6. Protocol specification for host  $u$

connectivity graph is linear with respect to the number of nodes in the group.

Figure 5 summarizes the state variables a node needs to keep for the execution of the protocol, and the support functions used in the protocol presentation that follows. A brief description of each function is included.

The protocol is presented in Figure 6. The table lists each action taken by host  $u$ , the action's precondition, and the action's effect, given the satisfaction of the precondition. There are two types of actions in the system. The first column of the figure shows actions that are triggered by a change in the local state at host  $u$ . The second column lists actions that are triggered by the arrival of the message at host  $u$ . Each of the actions in the latter group have the form GET MESSAGE. For each of these, there is a corresponding SEND MESSAGE. For example, GET NEIGHBORHELLO at host  $u$  is coupled with a SEND NEIGHBORHELLO at another host. The figure shows only the protocol executed at a single host,  $u$ , in the system. Each host in the network has its own instance of the actions shown.

Our implementation of the group membership maintenance protocol is discussed in the next section.

#### IV. IMPLEMENTATION

Our neighbor and group discovery protocol is built on top of the MAC layer, while the group membership service is built partly on an ad hoc routing layer. Fig. 7 shows the system architecture of our group membership service. The shaded areas indicate our contributed components.

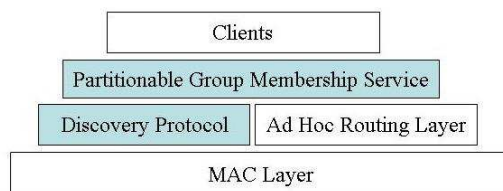


Fig. 7. System Architecture

The implementation of the service is written entirely in Java. The discovery layer employs a simple beacon based mechanism using a thread that periodically sends a hello

message. Another thread listens for incoming hello messages and passes this discovery information to the group membership layer. The main component in the group membership package is the `GroupMember` object, which contains several threads that control communication between the hosts in the network. Each different type of communication is handled by a different Java thread. These threads coordinate with each other through their owner object, the `GroupMember`. The coordination effort includes forwarding discovery information to the group leader, responding to merging and partitioning instructions, and updating the group leader with current location information. Group leaders carry the additional responsibilities of listening to their group members, communicating with other nearby group leaders, and periodically calculating the group's safety.

The group membership package presupposes ad hoc routing to be running on every host participating in the network. Therefore many of the messages discussed above are routed through other hosts in the network. As such, the leader of a group need not be directly connected to every member of the group.

The interface to the group membership protocol builds on the `EventObject` and `EventListener` classes in the Java language. An application running on a host that uses the group membership package to participate in groups in the network simply creates a `GroupMember` object. It then registers as a listener to `GroupChangedEvents` generated by its `GroupMember` object. When a new group configuration arises, the group membership package generates a `GroupChangedEvent` that is passed to all registered listeners. The application can take further actions, based on the implementation of this listener.

The `GroupMember` interface allows the user to specify the parameters needed for safe distance calculation. For example, the creator of the `GroupMember` can specify the host's maximum speed and its communication range. In addition to parameters for safe distance, the `GroupMember` creator also specifies the frequency of the hello beacon and the frequency of the group update messages to be sent to the leader.

While the implementation of the algorithm was a straightforward exercise in the use of Java threads and socket programming, some differences worth noting cause the implementation to vary from the examples presented in the previous sections. As presented, the protocol assumes that application level messages and the group membership protocol

messages are sent on the same channel. As indicated in the discussion on merging and partitioning, ensuring that messages are received in a FIFO order and that application messages are sent and received in the same group configuration requires some additional work. The example presented in the previous section used flush messages and configuration numbers to accomplish this. The implementation, however, attempts to separate as much as possible the group discovery and maintenance from the application level and therefore leaves the flush messages and configuration numbers presented as part of the example protocol to the particular application. This separation allows each application to choose its own mechanism for ensuring atomicity. Applications with weak consistency requirements may use the group membership package without any atomicity guarantees.

```

public class GroupMember implements GroupBeaconListener {
    public GroupMember(InetAddress leaderAdd, Location loc,
                      int period, int range, int maxSpeed,
                      int updatePeriod, int networkDelay);

    public void start();
    public void stop();
    public void pause();
    public void resume();
    public synchronized void addGroupChangedListener(GroupChangedListener gcl);
    public synchronized void removeGroupChangedListener(GroupChangedListener gcl);
    public void setLocation(Location newLocation);
    public void newGroupBeacon(GroupBeaconEvent gbe);
}

```

Fig. 8. The Public Interface of the Group Membership Package

Another concern addressed in the design was the clean separation between the group membership package and the application. By building on a model already integral to the Java language, the simple interface requires only that the application programmer understand the Java event model to successfully use the package. The simple interface composed of a single type of listener and a single type of event provides the desired ease of understanding. Figure 8 shows the public interface of the `GroupMember` object. The constructor accepts parameters for the safe distance calculation. With a handle to the `GroupMember` object, the programmer can start, stop, pause, and resume the `GroupMember` object. These methods affect the running of the threads that the `GroupMember` object uses

for communication. The programmer can also add and remove a `GroupChangeListener`. The two final methods are not used often by the application programmer as they are used by other packages necessary for the group membership protocol to function properly. The first method allows a location generating package (e.g., a GPS monitor) to update the physical location of the host. The second method allows the `GroupMember` to respond to beacon events that are generated by a separate beaconing package. These beacons are the multicast hello messages discussed previously. Figure 9 shows an example usage of the group membership package.

As indicated in the previous sections, this protocol was developed because the LIME [28], [25] middleware requires the ability to transparently and consistently share data across mobile hosts in ad hoc network environments. The LIME middleware as originally released required a mobile agent or host to explicitly announce its intention to engage or disengage from a group. The integration of this group membership protocol with the LIME middleware transforms the processes of engagement and disengagement into transparent reconciliations of LIME information when agents or hosts move in the network, thereby changing their status with respect to the protocol's safety requirements. In such a way, we were able to implement a mobility-aware version of LIME which automatically engages and disengages depending on the relative distances and mobility patterns of the participating hosts.

Because the group membership package is completely independent of LIME or any other application that may use it, changes to the package do not affect LIME, as long as the changes to the package do not affect its interface. This allows for future 'pluggable' versions of the group membership package to replace the current version. One can envision an implementation in which the safe distance is based on something more complex than physical location, such as signal strength, stability etc.

## V. SAFE DISTANCE ANALYSIS

The key feature of our algorithm is the use of location information and safe distance in the group membership management. The leader of a group frequently checks the members' locations to make sure that only those that are guaranteed to stay connected with the group for at least  $t + t'$  more units of time remain in the group, where  $t$  is the time

```

// The test class monitors the changes to a particular group member's group
// An instance of this class runs on each participating host
public class Test implements GroupChangeListener {
    // keep a handle to the group member object
    private GroupMember g;
    // integer count of the number of changes that have occurred
    private int changes = 0;
    public Test(GroupMember g) {
        this.g = g;
        // make this object a listener for events generated by the package
        g.addGroupChangeListener(this);
    }
    // this method is required by the GroupChangeListener interface
    // it is called when a new GroupChangedEvent occurs
    public void groupChanged(GroupChangedEvent gce) {
        // log the receipt of the change
        changes++;
        System.out.println("Change:" S + changes);
    }
    public static void main(String[] args) {
        // create a new GroupMember object for this host
        GroupMember g = new GroupMember(InetAddress.getLocalHost(),
                                         new Location(0,0), 1000, 3, 0, 100, 0);
        // create an instance of the Test class to monitor the GroupMember
        Test t = new Test(g);
        // start the GroupMember
        g.start();
    }
}

```

Fig. 9. An Example Use of the Group Membership Package

specified by the application layer and  $t'$  is the time bound for configuration changes. The combination of  $t$  and  $t'$  determines the safe distance for a specific operation, which could be the merging operation, the partitioning operation, or any other group operations specified by the application. Let's assume that  $t_d$  is the maximum delay between the time a control message is issued and the time it is received and processed, i.e., the sum of the maximum network delay and the maximum process queuing delays both at the sender and the receiver. For convenience, we refer to  $t_d$  as the network delay. In the case of splitting,

the maximum time it takes for a group to be partitioned successfully is twice the network delay.

If the leader continuously monitors the group configuration and all member locations are up to date, then mobility-induced unannounced disconnection can be caught in advance and dealt with successfully by requiring  $t' > 2 * t_d$ . Yet the leader's information about members' locations is always a little bit out of date. If members sample and report their locations every  $t_u$  units of time, the location information the leader has about a member could be outdated by time  $t_u + t_d$ . Taking this into consideration, the reserved time  $T$  must be greater than  $t_u + t_d + 2 * t_d = t_u + 3t_d$ . Whether or not we can use  $d_r = R - 2V_{max}(t_u + 3t_d)$  as the safe distance for partitioning depends on the requirement for merging. Because we do not allow a merging process to be aborted once committed, the computation of safe distance for partitioning also needs to account for the time associated with the merging process. Consider the following scenario: immediately before a commit in a merging process, the group configuration is safe using safe distance  $d_r$ ; right after the commit, a leader might discover that its group is no longer safe, and a partition process needs to be carried out immediately. However, the merging process hasn't finished. This is not acceptable. Taking into account that the two-phase merging process needs at most an execution time of  $4t_d$  (4 messages), and the configuration needs to be safe right after merging, the total reserved time for both merging and partitioning needs to be  $t_u + 3t_d + 4t_d = t_u + 7t_d$ . In other words, the safe distance for both merging and partitioning is

$$d_s = R - 2V_{max}(t_u + 7t_d) \quad (2)$$

Using the same distance for merging and partitioning introduces the problem of 'shuttle nodes', i.e., if a node is moving in and out the safe boundary, mergers and partitions occur repeatedly. To avoid this, one can further tighten the safe distance for merging, creating a 'buffer zone', and thus reducing the probability of shuttling.

Our algorithm also requires  $V_{max}$  to be no greater than  $V_{adm}$ , i.e., the maximum admissible speed for the specific wireless network system the mobile hosts are using. Most wireless network systems (e.g. DECT, GSM, PCS, ETACS) have a maximum admissible

speed [9]. When a mobile node is moving too fast, it simply becomes invisible to the network. For GSM and PCS,  $V_{adm}$  is about  $50m/s$ ; for DECT microcellular system,  $V_{adm}$  is about  $11m/s$ . Without the condition  $V_{max} \leq V_{adm}$ , a speed change from  $V \leq V_{adm}$  to  $V > V_{adm}$  creates an unannounced disconnection. Speed monitoring would be needed to prevent this kind of unannounced disconnection from happening.

Figure 10 illustrates the relation between the safe distance  $r$  and the maximum admissible network delay  $t_d$  with reasonable values of  $R = 150m$ ,  $V_{max} = 10m/s$  and location reporting frequency of  $1 Hz$  ( $t_u = 1s$ ). One can see that as the delay bound increases, the safe distance decreases.

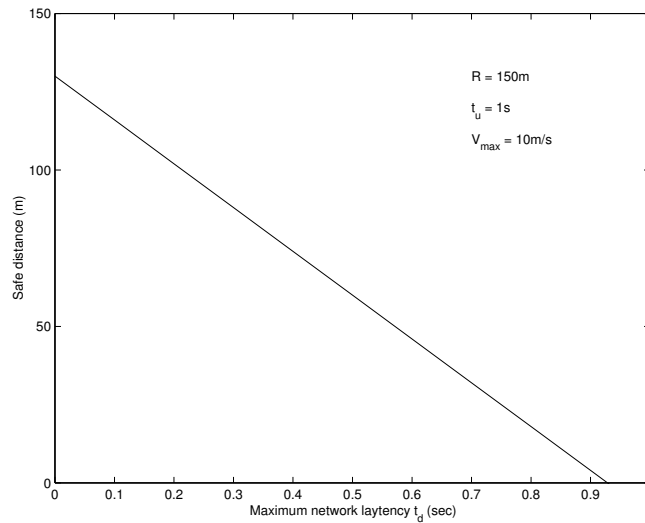


Fig. 10. Safe distance vs. network delay

Figure 11 shows the relation between the safe distance threshold, the upper bound on speed, and the network delay bound. The region above the top curve corresponds to  $d_s < 0$ . In this parameter space we cannot provide any consistency guarantees for a group containing more than one member. On the other hand, if a mobile system's network delay bound and maximum speed bound fall into the region below the ( $d_s = 90m$ ) curve, we could provide the group view consistency guarantee by using  $90m$  as maximum safe distance.

Choosing the proper safe distance is key to the protocol. A choice that is not conservative enough might endanger the correctness of the group membership service. A choice that is

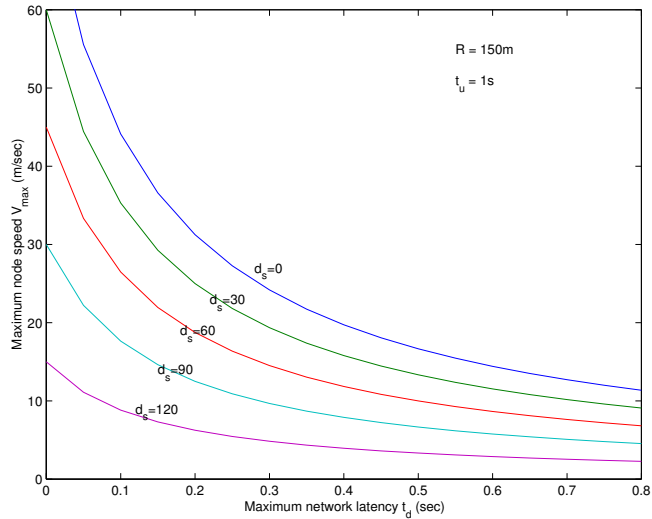


Fig. 11. Relation between safe distance, speed bound and delay bound

too conservative might cause the groups to be too small to be useful for some applications, or smaller than necessary. One must strike a delicate balance between these conflicting choices to make the group membership service as useful as possible.

## VI. DISCUSSION

Our partitionable group membership specification is stronger than traditional ones in that it not only requires availability during partition but also emphasizes consistency during partition. Previous work in group membership either admits inconsistency during the partition, or reduces availability during partition.

On the other hand, the strong properties required by our partitionable group membership specification make it impossible to implement in traditional asynchronous system models. Key to our solution to the strong partitionable group membership problem is the notion of safe distance and the corresponding notion of a logical connectivity graph. Given information about physical properties of the mobile system, we are able to predict certain behavior of the system. In turn, we are able to achieve the strong consistency required by the group membership service.

At present, our algorithm assumes that all mobile nodes in the system have a known maximum speed. Unbounded speed is another possible source of unannounced disconnec-

tion. Low speed is a requirement for most wireless networks. In systems involving mobile nodes that can control their own velocity, e.g., cars and planes, a safe relative velocity threshold can be used in the decision of merging and splitting. Of course, in such cases we would have to assume a maximum acceleration for the mobile nodes in order to make disconnection predictions possible.

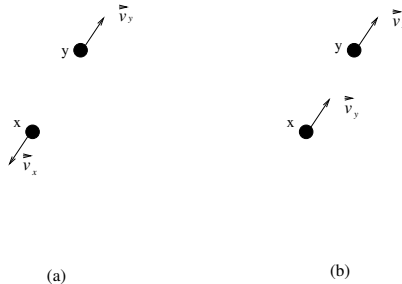


Fig. 12. Contribution of velocity information

Our membership service can be further refined when velocity information about each mobile host is available. For example, consider cases (a) and (b) of Figure 12. In case (a), hosts  $x$  and  $y$  are moving away from each other, while in (b), they are moving in the same direction. Clearly,  $x$  and  $y$  are less likely to disconnect in the latter case than in the former. Translating this into the language of safe distance, the maximum safe distance between  $x$  and  $y$  is greater in case (b) than in case (a). In the current algorithm, as we assume the velocity information is not available. Since we cannot differentiate cases (a) and (b), we have to consider the worst case movement scenario for each pair of hosts, i.e., they may be moving away from each other at the maximum relative speed at any point in time. When velocity information is available, the safe distance threshold between hosts  $x$  and  $y$  (in Figure 12) can change dynamically according to the formula:

$$R - |\vec{v}_x - \vec{v}_y| \cdot t - |\vec{a}_{max} \cdot t^2| \quad (3)$$

where  $\vec{a}_{max}$  is the maximum acceleration for all hosts and  $t$  is the time needed for a group operation already in progress to finish. We can change our algorithm to use the velocity information in the following way: (1) each host includes its velocity information in hello messages and location-update messages, and (2) the safe distance is computed using Equation (3) with  $t = t_u + 7t_d$ . The rest of the algorithm remains the same.

Although only safe physical distance is employed in our protocol to avoid unannounced disconnection, other physical attributes can also be used to determine safety. For instance, if link failure is predictable through monitoring the bandwidth or transmission power change between two nodes, a similar group membership protocol can be built by exploiting similar concepts, e.g., ‘safe bandwidth’ or ‘safe power level’ etc.

We assume each mobile host has knowledge of its own location. This is made possible by the availability of positioning systems such as GPS. Yet positioning systems are not always accurate. For simplicity we did not consider this in our safe distance analysis. One can always factor data precision and sample frequency of a location system into the safe distance and make the service more robust.

We assume that the model of wireless communication has a known communication radius. This choice, often quoted as the “unit disk model” ([30],[31]), is widely used in theoretical studies about the properties of wireless ad hoc networks. While this simplistic radio communication model is an appropriate starting point for reasoning about certain wireless network properties in an open environment, such as highway (automobiles) and desert (battle groups), it is not valid for indoor scenarios. The reason is that radio communication has very unpredictable characteristics in the indoor environment (due to reflection, absorption, interference, etc.), and an omnidirectional range is not a good abstraction. In turn, our safe distance based group membership service is applicable only to open field environments in which the omnidirectional wireless communication range is an appropriate approximation.

The correctness of our algorithm relies on the assumption that the network has a delay bound. At this moment, we are not aware of any ad hoc routing protocols which can provide a good delay bound. Yet, it is conceivable that a routing protocol with good delay bound for prioritized group control messages is possible by restricting group size and using location information. An alternative approach is to augment the merging criteria with a maximum group size or even some group spatial configuration condition. By doing this, it may be possible to meet the delay bound assumption with high probability.

## VII. CONCLUSION

The motivation for this work rests with our desire to provide data consistency in applications that execute over ad hoc networks. Yet, maintaining a consistent view of the global state in a distributed network is difficult in general and essentially impossible in the presence of unannounced disconnections. In ad hoc mobile systems, mobility-induced unannounced disconnection occurs frequently, as part of the normal operation of the network. This makes the development of fault-tolerant systems on top of ad hoc networks very challenging. Our goal of assisting software developers in their efforts to design and build reliable mobile applications leads us to define a new partitionable group membership service with strong consistency requirements. We have also presented a strategy and an algorithm to implement the service, given appropriate system assumptions. The novel feature of the algorithm is its ability to create the illusion of announced disconnection. By using location and mobility information about the mobile hosts in the region, the membership service is able to guarantee to the application layer a reliable message delivery service to group members in the presence of mobility-induced unannounced disconnection, given appropriate system assumptions. This approach represents a new direction in fault-tolerant distributed computing, one that factors into protocols information about mobility and space. This work also provides a practical solution to masking mobility induced unannounced disconnections in ad hoc mobile systems.

## ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation under Grant No. CCR-9970939. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors thank Ali Hazemi and Amy Murphy for helpful discussions during the development of these algorithms. The authors also thank the reviewers for their thoughtful suggestions. A preliminary report on this work appeared in *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, May 2001 [32].

## REFERENCES

- [1] F. Cristian, "Reaching agreement on processor-group membership in synchronous distributed systems," *Distributed Computing*, vol. 4, no. 4, pp. 175–188, 1991.
- [2] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "Transis: A communication subsystem for high availability," in *FTCS-22: 22nd International Symposium on Fault Tolerant Computing*, Boston, Massachusetts, 1992, pp. 76–84, IEEE Computer Society Press.
- [3] K. P. Birman, "The process group approach to reliable distributed computing," *Communications of the ACM*, vol. 36, no. 12, pp. 37–53, December 1993.
- [4] A. Ricciardi and K. Birman, "Process membership in asynchronous environments," Tech. Rep. 93-1328, Cornell University, Department of Computer Science, February 1993.
- [5] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, "The Totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems*, vol. 13, no. 4, pp. 311–342, 1995.
- [6] P. D. Ezhilchelvan, R. A. Macedo, and S. K. Shrivastava, "Newtop: A fault-tolerant group communication protocol," in *Proceedings of the International Conference on Distributed Computing Systems*, 1995, pp. 296–306.
- [7] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and Colleen A. Lingley-Papadopoulos, "Totem: A fault-tolerant multicast group communication system," *Communications of the ACM*, vol. 39, no. 4, pp. 54–63, 1996.
- [8] R. van Renesse, K.P. Birman, and S. Maffeis, "Horus, a flexible group communication system," *Communications of the ACM*, vol. 39, no. 4, pp. 76–83, April 1996.
- [9] R. Prakash and R. Baldoni, "Architecture for Group Communication in Mobile Systems," in *Proceedings of the IEEE Symposium on Reliable Distributed Systems (SRDS)*, October 1998, pp. 235–242.
- [10] O. Babaoglu, R. Davoli, and A. Montresor, "Group communication in partitionable systems: Specification and algorithms," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 308–336, April 2001.
- [11] G. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: A comprehensive study," *ACM Computing Survey*, vol. 33, no. 4, pp. 427–469, December 2001.
- [12] European Transport Safety Council, *Intelligent Transportation Systems and Road Safety*, Brussels, 1999.
- [13] ITS, "Problem Area Descriptions, Motor Vehicle Crashes - Data Analysis and IVI Program Emphasis," USDOT and ITS Joint Program, November 1999.
- [14] Peter Seiler, Bongsob Song, and J. Karl Hedrick, "Development of a Collision Avoidance System," in *SAE Conference Proceedings*, 1998, pp. 97–103.
- [15] Qingfeng Huang, Ronald Miller, Perry McNeille, David Dimeo, and Gruia-Catalin Roman, "Development of a peer-to-peer collision warning system," *Ford Technical Journal*, vol. 5, no. 2, March 2002.
- [16] Zhengrong Yang, Takashi Kobayashi, and Tsuyoshi Katayama, "Development of an Intersection Collision Warning System Using DGPS," in *Intelligent Vehicle Systems(SP-1538)*. March 2000, SAE World Congress, Detroit, Michigan.
- [17] Shannon Hetrick, "Examination of Driver Lane Change Behavior and The Potential Effectiveness of Warning Onset Rules for Lane Change Or "Side" Crash Avoidance Systems," M.S. thesis, Virginia Polytechnic Institute and State University, March 1997.
- [18] A. Fekete, N. Lynch, and A. Shvartsman, "Specifying and using a partitionable group communication service," *ACM Transactions on Computer Systems*, vol. 19, no. 2, pp. 171–216, 2001.

- [19] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg, “On the formal specification of group membership services,” Tech. Rep. 95-1534, INRIA, France, 1995.
- [20] D. Dolev, D. Malki, and R. Strong, “A framework for partitionable membership service,” in *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, May 1996.
- [21] D. Dolev, D. Malki, and R. Strong, “An asynchronous membership protocol that tolerates partitions,” Tech. Rep. 94-6, Institute of Computer Science, The Hebrew University of Jerusalem, 1994.
- [22] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” *Journal of ACM*, vol. 32, no. 2, pp. 374–382, 1985.
- [23] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost, “On the impossibility of group membership,” in *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC’96)*, New York, USA, 1996, pp. 322–330, ACM.
- [24] F. Cristian and C. Fetzer, “The timed asynchronous system model,” Tech. Rep. CS-97519, University of California - San Diego, January 1997.
- [25] A. L. Murphy, G. P. Picco, and G.-C. Roman, “Lime: A middleware for physical and logical mobility,” in *Proceedings of the 21st International Conference on Distributed Computing Systems*, 2001, pp. 521 – 533.
- [26] A. L. Murphy, G. P. Picco, and G.-C. Roman, “Lime: A coordination middleware supporting mobility of agents and hosts,” Tech. Rep. WUCSE-03-21, Washington University Department of Computer Science and Engineering, St Louis, 2003.
- [27] F. Cristian, “Synchronous and asynchronous group communication,” *Communications of the ACM*, vol. 39, no. 4, pp. 88–97, 1996.
- [28] G.P. Picco, A.L. Murphy, and G.-C. Roman, “LIME: Linda meets mobility,” in *Proceedings of the 21<sup>st</sup> International Conference on Software Engineering (ICSE)*, May 1999, pp. 368–377.
- [29] F. Cristian, “Group, majority, and strict agreement in timed asynchronous distributed systems,” in *Proceedings of the 26<sup>th</sup> International Symposium on Fault-Tolerant Computing*, 1996, pp. 178–187.
- [30] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu, “Geometric spanner for routing in mobile networks,” in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Long Beach, CA, USA, 2001, pp. 45–55.
- [31] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger, “Geometric ad-hoc routing: of theory and practice,” in *Proceedings of the 22<sup>nd</sup> ACM International Symposium on the Principles of Distributed Computing (PODC)*, Boston, Massachusetts, USA, 2003.
- [32] G.-C. Roman, Q. Huang, and A. Hazemi, “Consistent group membership in ad hoc networks,” in *Proceedings of the 23<sup>rd</sup> International Conference on Software Engineering (ICSE)*, May 2001.