Localized Group Membership Service for Ad Hoc Networks

Linda Briesemeister
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025, U.S.A.
linda.briesemeister@sri.com

Günter Hommel
Technical University of Berlin
Einsteinufer 17
10587 Berlin, Germany
hommel@cs.tu-berlin.de

Abstract

We present a specification for a new, localized group membership service that maintains the membership status of adjacent nodes – called neighbors – in a mobile distributed system. The service builds on top of a neighborhood service which employs a simple heartbeat mechanism to discover and track neighbors in the mobile network. Both services assume unreliable communication as found in the wireless environment. No knowledge of the network topology is presumed.

We impose a deadline for installing views of the membership to force timely deciding protocols. We give a simple implementation of the neighborhood and the group membership service. If the deadline of view installations is at least the heartbeat rate, we can prove the correctness of our suggested implementation. An application in mobile ad hoc networking exemplifies potential areas of deployment for a localized group membership service.

1. Introduction

The group communication paradigm [4] embodies a prominent technique in fault-tolerant and reliable distributed computing. Groups of member processes therein interact and communicate in order to achieve a common goal. A group communication system integrates a group membership service with a reliable multicast service. The task of the group membership service is to keep members consistently informed about the current membership of a group by installing views. Processes can join and leave the group or even crash – all resulting in dynamic changes of the membership. Installed views consist of a set of members and reflect the perception of the group's membership. This requires the members to agree on the composition of a view.

In this article, we develop a group membership service for applications in distributed systems with mobile hosts. The distributed system model matches a mobile ad hoc network, in which hosts are moving freely and communicate through wireless links while in transmission range.

Two prominent characteristics distinguish our model from common distributed systems. First, we allow an unbounded number of processes to exist concurrently. None of the hosts are aware of an upper threshold on population size. Second, distributed applications in the context of mobile hosts are prone to temporary disconnections. Thus, recent work in group membership specification relaxes the demand for agreement on a single view and allows multiple disjoint views to exist concurrently in different network components.

We extend this idea and propose reducing the membership problem to the local environment of a host to cope with the severe conditions inherent to mobile ad hoc networks. A localized group membership service (LGMS) tracks the membership only of the adjacent neighbors. Changes in the localized group membership – existent neighbors join or leave the group voluntarily or crash, new members move into vicinity – are installed as local views at each host. These views differ according to the neighborhood relation among hosts.

2. Related Work

In recent years, several approaches to group communication and to building fault-tolerant toolkits have been reported, including Transis [14], Ensemble [7], Newtop [8], and Jgroup [10]. For group membership, Chandra et al. [6] have proven the impossibility of providing a group membership service in asynchronous systems with crash failures. However, this result strictly applies to primary-partition membership services, which allow only one network component – called the primary component or partition – to continue running the service, whereas processes in other networks are considered faulty. In contrast, a partitionable membership service relaxes the rigorous demand of delivering the same sequence of views to all members and allows multiple disjoint views to exist concurrently in different net-

work components. In the context of mobile ad hoc networks, temporary disconnections occur frequently. Hence, the service should be partitionable and therefore escapes the impossibility proof. However, no final agreement yet exists about a general specification of a *partitionable* group membership service.

Keidar et al. [11] study the task of group membership in the context of wide area networks. Here, the membership service resides on dedicated servers which are not involved in the communication among the group. This approach makes the service scalable both in terms of the number of groups and in the number of members in each group.

An interesting core idea of the wide area membership service is to avoid delivering obsolete views. The membership service waits for agreement among all the view members about what the view should be. It neither delivers a view without such agreement nor does it deliver an obsolete view when it has new information that the membership has changed. This implies that the algorithm may not terminate if the network cannot stabilize fast enough. On the other hand, this policy avoids network congestion caused by control messages dealing with an outdated view.

The idea of waiting at the expense of not deciding in an unstable environment is appealing in the context of ad hoc networks where message overhead in an inherent unreliable communication scenario needs to be carefully observed. However, due to constant changes in highly mobile ad hoc networks, the stabilization period is short and a time limit must prevent the membership service from waiting forever. Also, the inherent hierarchy in client/server approaches and predefined, dedicated servers do not exist in mobile ad hoc networks. Therefore, we cannot apply this membership service in our environment.

At the University of Bologna, the group communication paradigm has been studied and implemented for example in the Jgroup [10] project. Advances in "partition-aware" group communication systems are reported in [3, 2, 13]. "Partition-aware" applications continue operating without blocking when the network fragments and reconfigure themselves when partitions merge. Babaoğlu et al. [2] specify a partitionable group membership service that guarantees liveness and excludes trivial solutions. They give an implementation that satisfies the specification in distributed systems with a certain stability.

3. System Model

To specify our system model and properties, we use the notation of Manna and Pnueli's temporal logic [12]. The time is linear and discrete, starting from an initial point. Thus, the past time operators can at most reach back until this starting point. We derive the bounded operators from Henzinger et al. [9]. Table 1 gives an overview of the temporal operators.

Table 1. Overview on temporal logic operators.

$\Diamond_{\leq k}$	bounded "eventually", "sometimes" (within next k time steps)
$P \,\mathcal{U}\, Q$[1]	"until" (Q happens eventually and until then P holds)
$P \,\mathcal{S}\, Q$	"since" (Q happened before and since then P holds)

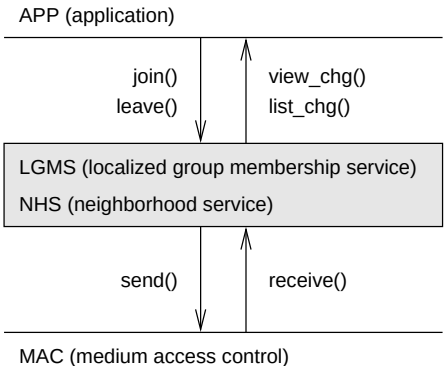

Figure 1. Architecture of a process.

The asynchronous distributed system consists of processes p which communicate solely via messages sent through a radio channel. Exactly one process exists in every node of the ad hoc network – therefore, we do not distinguish processes from nodes or hosts. Processes have a unique identifier p, $(p \in \mathbb{N})$ of which they are aware. There is no common clock or common memory, and the relative speed of processes is undetermined.

Each process executes by performing events from a finite set S of valid events sequentially. The communication between processes is modeled by send() and receive() events. A process interacts with its application by join(), leave(), list_chg(), and view_chg() events. While a process is idle it performs the null event ϵ. The function σ captures the sequence of events performed by every process. Figure 1 provides an overview of the architecture of a process.

Processes can fail by crashing permanently at any time of execution. After a process has crashed, it performs the dead event δ. Also, processes may start later than the beginning of the global time. From $t = 0$ until a process is started for the first time ever, it also performs dead events δ. We call these processes and the crashed processes inactive, whereas processes are active if they are idle or if they perform events from the set of valid events S. Processes are not aware of the point in time that they fail i.e. an active process cannot determine when it will crash. Other than crash failures, the nodes in the distributed system behave

[1]Note, that if Q never happens, then $P \,\mathcal{U}\, Q$ is false even if P holds infinitely.

benignly. In particular, we do not consider arbitrary, malicious, or Byzantine faults like sending spurious messages or exhibiting any unpredictable behavior.

There is no assumption on the underlying network topology. It is unlikely that all nodes of the mobile ad hoc network are in transmission range of each other, hence the network structure is not fully connected. Furthermore, we cannot even assume that the imposed network graph is connected all the time i.e. a path between every pair of nodes exists. In case part of the network becomes disconnected from another part, the network is called fragmented or partitioned. Processes are neither aware of the other processes nor of the momentary network topology unless they draw conclusions from the messages they have received lately.

Each process is associated with a location in time and space and may travel during operation. Depending on the location of nodes in space, wireless communication is possible when nodes are in transmission range of each other. The transmission delay while broadcasting a radio message is negligible i.e. in our model, the message will be received immediately. The radio channel is unreliable and thus messages can be lost; however, if the message is received then we assume it to be correct. Every reception of messages must be caused by someone sending the message.

A variety of reasons can inhibit communication over wireless links. In our model, communication failures are grouped into send and receive omission. A send omission happens for example if the underlying medium access control (MAC) fails in claiming the channel for transmission. Packet loss can occur in presence of strong multipath fading or because of shadowing effects if the chosen frequency demands line of sight. Additionally, atmospheric dilution and hidden stations may cause packet errors. We say that a receiving host suffers from a receive omission fault, if a packet is lost.

Refer to Briesemeister [5] for more details about the proposed system model.

4. Neighborhood Service

We propose to employ a service that yields a list of adjacent processes which are expected to be alive at the moment. This neighborhood service is based on a simple heartbeat mechanism that repeatedly beacons the own process identity to its neighbors with a fixed rate τ_{hb}. The neighborhood service of a process collects the heartbeats from other adjacent processes and maintains a list of current neighbors.

Based on the heartbeats, we introduce the concept of a process being connected to another process. Note, that this concept is inherently asymmetric; a process that receives consecutive heartbeats suspects another process to be its neighbor – no assumption is made on how the other process perceives that process. The beginning of a connection is the first reception at a process p of a heartbeat from a process q after the duration of at least the heartbeat rate τ_{hb} in which the process p has not received a heartbeat from q. We call q then "newly connected" to p; in short $q \triangleright p$. The end of a connection equals the time out of waiting for the next heartbeat. We say that if such a time out occurs, q is "disconnected" from p; in short $q \triangleleft p$. During the interval between q is newly connected and disconnected from p we say that q is "transiently connected" to p; in short $q \bowtie p$.

The relation "disconnected" is not complementary to the relation "transiently connected." Two processes can be neither transiently connected nor disconnected. Also, if a process gets disconnected from another process then they can still be in vicinity. For example, one of the processes can suffer from a send or receive omission failure during the heartbeat.

In our model, the function $list(p)$ applied to a process p yields the list of neighbors that p has installed through the last list_chg() event. If p has not performed a list_chg() event yet or is crashed then $list(p)$ returns an empty set.

Now, we use the notation introduced above to specify the neighborhood service (NHS). The service should react with list_chg() events if the process gets newly connected or disconnected from another process. Also, the list reported to the upper layer must be accurate and complete such that it only includes those processes assumed to be neighbors from which it has recently received heartbeats.

NHS 1 (New Neighbors) *If process q is newly connected to process p at time t then p performs at least once a list_chg() event within the next $\tau_{hb} - 1$ time steps or p crashes. Formally,*

$$q \triangleright p \Rightarrow \Diamond_{\leq \tau_{hb}-1} (\sigma(p) = \textit{list\_chg()} \vee \sigma(p) = \delta)$$

NHS 2 (Leaving Neighbors) *If process q is disconnected from process p at time t then p performs at least once a list_chg() event within the next $\tau_{hb} - 1$ time steps or p crashes. Formally,*

$$q \triangleleft p \Rightarrow \Diamond_{\leq \tau_{hb}-1} (\sigma(p) = \textit{list\_chg()} \vee \sigma(p) = \delta)$$

NHS 3 (Accuracy) *If process p installs a list and q is transiently connected to process p then the installed list includes q. Formally,*

$$\sigma(p) = \textit{list\_chg()} \wedge q \bowtie p \Rightarrow q \in list(p)$$

NHS 4 (Completeness) *If process p installs a list and q is not transiently connected to process p then the installed list excludes q. Formally,*

$$\sigma(p) = \textit{list\_chg()} \wedge \neg q \bowtie p \Rightarrow q \notin list(p)$$

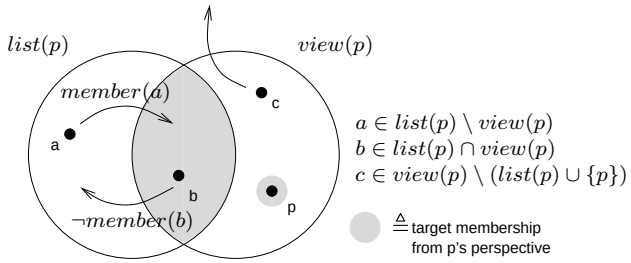

Figure 2. Three situations that require view changes at member p.

5. Localized Group Membership Service

For the sake of brevity, we assume that only one group exists in each run to omit group identifiers. This implies that in the case of multiple groups the characteristics of a group can be communicated as a small description of parameters such that processes can distinguish them. Processes decide upon local parameters for their own membership. The application layers issues join() and leave() events to its own group membership layer. For a meaningful group membership service, we assume that the application layer of an active process always alternates the join() and leave() events starting with the join() event. Then, the boolean function $member(p)$ applied to a process p is true, if and only if p has performed the join() event before and since then, p neither performed the leave() event nor crashed.

At each member process, the localized group membership service (LGMS) tracks the group membership of the adjacent neighbors. A process installs changes in the localized group membership as views through the view_chg() event. A view reflects the current situation of the membership from the perspective of a certain process. The function $view(p)$ applied to a process p yields the set of process identifiers that p has installed through the last view_chg(). If p has not installed a view, the set is empty.

We introduce a timing value τ_{vc} for the view to change. After a process performs a join() and leave() event in the LGMS layer, the process must react within the next τ_{vc} time steps by installing a new view through the view_chg() event. Other situations in which a member process has to install a new view are drawn as a set diagram in Figure 2. There, three constellations require a member process p to adjust its view: If neighbor a becomes a member, p must include a into its view. If neighbor b leaves the group, p must remove b from its view. Finally, c must be excluded from p's view because $c \notin list(p)$ and thus c is not a neighbor of p.

Now, we define the properties LGMS 1–6 of the sketched localized group membership service. We make use of the concepts introduced above and the neighborhood service that reports changes in the neighborhood through the list_chg() event.

LGMS 1 (View Integrity) *(i) Every view installed at member p includes the process itself. Formally,*

$$member(p) \wedge \sigma(p) = \textit{view\_chg()} \Rightarrow p \in view(p)$$

(ii) Every view installed at non-member p is empty. Formally,

$$\neg member(p) \wedge \sigma(p) = \textit{view\_chg()} \Rightarrow view(p) = \emptyset$$

LGMS 2 (Limit on Neighborhood) *Only neighbors are part of a view installed at member p. Formally,*

$$member(p) \wedge \sigma(p) = \textit{view\_chg()} \Rightarrow view(p) \subseteq list(p) \cup \{p\}$$

LGMS 3 (View Accuracy) *If member p has another member q in its view, then q remains in p's view until q is not a neighbor or p or q leaves the group or p or q crashes. Formally,*

$$\begin{aligned} member(p) \wedge \exists q \in view(p) \setminus \{p\} : member(q) \Rightarrow \\ q \in view(p) \, \mathcal{U} (q \notin list(p) \vee \\ \neg member(p) \vee \neg member(q) \vee \\ \sigma(p) = \delta \vee \sigma(q) = \delta) \end{aligned}$$

LGMS 4 (View Completeness) *If member p has a neighbor q which is not in p's view nor a member, then q is excluded form p's view until q is not a neighbor anymore or q becomes a member or p leaves the group or q crashes. Formally,*

$$\begin{aligned} member(p) \wedge \exists q \in list(p) \setminus view(p) : \neg member(q) \Rightarrow \\ q \notin view(p) \, \mathcal{U} (q \notin list(p) \vee \\ member(q) \vee \neg member(p) \vee \\ \sigma(p) = \delta \vee \sigma(q) = \delta) \end{aligned}$$

LGMS 5 (View Installation) *(i) If process p joins or leaves the group, it installs a new view within the next τ_{vc} time steps or p crashes. Formally,*

$$\begin{aligned} \sigma(p) \in \{\textit{join()}, \textit{leave()}\} \Rightarrow \\ \Diamond_{\leq \tau_{vc}} (\sigma(p) = \textit{view\_chg()} \vee \sigma(p) = \delta) \end{aligned}$$

(ii) If neighbor q of a member p is a member but not included in p's view, then p includes q in its view within the next τ_{vc} time steps or q is not a neighbor anymore or p or q leaves the group or p or q crashes. Formally,

$$\begin{aligned} member(p) \wedge \exists q \in list(p) \setminus view(p) : member(q) \Rightarrow \\ \Diamond_{\leq \tau_{vc}} (q \in view(p) \vee q \notin list(p) \vee \\ \neg member(q) \vee \neg member(p) \vee \\ \sigma(p) = \delta \vee \sigma(q) = \delta) \end{aligned}$$

(iii) If neighbor q included in member p's view is not a member, then q is excluded from p's view within the next τ_{vc} time steps or q becomes a member and a neighbor. Formally,

$$member(p) \wedge \exists q \in list(p) \cap view(p) : \neg member(q) \Rightarrow$$
$$\Diamond_{\leq \tau_{vc}} (q \notin view(p) \vee (q \in list(p) \wedge member(q)))$$

(iv) If process q included in member p's view is not a neighbor, then q is excluded from p's view within the next τ_{vc} time steps or q becomes a neighbor and a member. Formally,

$$member(p) \wedge \exists q \in view(p) \setminus (list(p) \cup \{p\}) \Rightarrow$$
$$\Diamond_{\leq \tau_{vc}} (q \notin view(p) \vee (q \in list(p) \wedge member(q)))$$

A group membership service usually requires that if certain events occur, then a new view of the group must be eventually installed to reflect the changes. In our model, we capture this in LGMS 5 where a view change is required after a certain time limit τ_{vc}. As pointed out in [1], another requirement should then prevent capricious view changes, namely that a new view is installed *only* if certain events previously occurred. In our specification, we add LGMS 6 to overcome this problem.

LGMS 6 (View Justification) *If process p installs a new view, one of the triggering events from the view installation property LGMS 5, (i)–(iv) happened before and since then no new view has been installed at p. Formally,*

$$\sigma(p) = \textit{view\_chg()} \Rightarrow (\sigma(p) \neq \textit{view\_chg()} \, \boldsymbol{\mathcal{S}} \, P)$$

where P is replaced by

$$P \leftarrow \sigma(p) \in \{\textit{join()}, \textit{leave()}\} \vee$$
$$(member(p) \wedge \exists q \in list(p) \setminus view(p) : member(q)) \vee$$
$$(member(p) \wedge \exists q \in list(p) \cap view(p) : \neg member(q)) \vee$$
$$(member(p) \wedge \exists q \in view(p) \setminus (list(p) \cup \{p\}))$$

6. Implementation

In this section, we present an implementation to solve NHS and LGMS. Then, we prove the correctness of our algorithm, if the deadline τ_{vc} to install views is at least as long as the heartbeat rate τ_{hb}.

The proposed implementation simply adds the current membership status as a boolean value to the heartbeat message that processes send periodically. Upon receiving heartbeat messages from other processes, a process maintains the list of neighbors and the view. The pseudo-code is printed below. We highlighted those lines in the code that cause a view installation for easier reference in the proofs. We assume that every sequence under "init" or "upon" is executed within one time step.

```
init
 1: list() ← ∅, view() ← ∅
 2: member() ← FALSE
 3: set(ownTimer,τ_hb)
 4: set(otherTimers(·),0) // unset all other timers
upon join()
 5: member() ← TRUE
 6: view() ← view_chg({ownID}) // include myself into view
upon leave()
 7: member() ← FALSE
 8: view() ← view_chg(∅) // clear view
upon ownTimer expires
 9: send(ownID,member())
10: set(ownTimer,τ_hb)
upon otherTimers(ID) expires
11: // remove from list of neighbors
12: list() ← list_chg(list()\ {ID})
13: // remove from view
14: if member() then // only modify view if member
15:    if ID ∈ view() then
16:       view() ← view_chg(view()\ {ID})
17:    end if
18: end if
upon receive(ID,status)
19: // neighborhood update
20: if ID ∉ list() then
21:    list() ← list_chg(list()∪ {ID})
22: end if
23: // membership update
24: if member() then // only modify view if member
25:    if status then // heartbeat came from member
26:       if ID ∉ view() then
27:          view() ← view_chg(view()∪ {ID})
28:       end if
29:    else // heartbeat came from non-member
30:       if ID ∈ view() then
31:          view() ← view_chg(view()\ {ID})
32:       end if
33:    end if
34: end if
35: // reset timer
36: set(otherTimers(ID),τ_hb)
```

First, we show that for any $\tau_{vc} < \tau_{hb}$ the implementation does not guarantee the specification. Construct a case with only two processes p and q in vicinity of each other. At t_1, p is a member and receives a heartbeat from non-member q. Then, $q \in list(p) \setminus view(p)$ and p sets otherTimers(q) to $t_4 := t_1 + \tau_{hb}$. Assume q joins at $t_2 := t_1 + 0.5 \cdot (\tau_{hb} - \tau_{vc})$ so that at t_2 all requirements of LGMS 5 (ii) hold. Set $t_3 := t_2 + \tau_{vc}$, which is smaller than t_4. However, p learns about q's membership not earlier than t_4, when

the next heartbeat is scheduled, but at t_3 the conjunction of $q \notin view(p)$, $q \in list(p)$, $member(q)$, $member(p)$, $\sigma(p) \neq \delta$, and $\sigma(q) \neq \delta$ holds. This violates LGMS 5 (ii), which requires p to include q into its view until t_3. □

Therefore, the deadline τ_{vc} must be equal or greater than τ_{hb}. With this assumption, we prove the properties LGMS 1–6 in the remaining part of this section.

View Integrity: (i) If a process is a member, than it has performed join() and since then not performed leave(). After joining, our algorithm installs a view that contains only the own identity (line 6.) Views that are installed later while being a member (lines 16, 27, 31) never remove the own identity from the view assuming that the identity of the neighbor (ID) is different from the own. □ (ii) Upon initialization, every process is a non-member and its view is by definition empty until it performs join(). After this, a process is a non-member, if it has performed leave() and since then not performed join(). When a process performs leave(), it installs an empty view (line 8.) Other, non-empty views are only installed (lines 6, 16, 27, 31) if the process is a member. □

Limit on Neighborhood: If a process is a member, than it has performed join() and since then not performed leave(). Assume an arbitrary $q \in view(p)$. Then, two cases hold. Either, $q = p$ and it is easy to see $q \in list(p) \cup \{p\}$. Or, $q \neq p$. In this case, q was inserted into $view(p)$ once before through a receive(q,TRUE) event at line 27. Now distinguish two cases depending on the time before: If $q \in L$ then $q \in list(p)$. If $q \notin L$ before the process performs receive(q,TRUE), then it includes q at line 21 such that $q \in list(p)$. □

View Accuracy: Let process p be a member with another member $q \in view(p)$. There are three cases in which q gets deleted from p's view. First, if p leaves the group, an empty view is installed (line 8.) It follows that $q \notin view(p)$ anymore and $\neg member(p)$. Second, the timer for q expires. Then, q is deleted from $view(p)$ (line 16.) In the same procedure, q is extracted from $list(p)$ (line 12.) Third, q is removed from $view(p)$ (line 31) when p performs receive(q,FALSE) and $q \in view(p)$ before. This means, that q has left the group and $\neg member(q)$. □

View Completeness: Let process p be a member with a non-member $q \in list(p) \setminus view(p)$. There is one case in which p includes q into its view (line 27): If member p performs receive(q,TRUE) then q is a member and it is included in p's list. □

View Installation: (i) When a process performs join(), it immediately installs a view (line 6.) □ (ii) Let process p be a member with a member $q \in list(p) \setminus view(p)$. Then, the last heartbeat from q received by p happened before at a time $t_1 > \text{now} - \tau_{hb}$. Otherwise, q would not be in p's list of neighbors. Also, q was not a member at t_1, because it was not included in p's view then. If neither of p and q crashes or leaves the group, and if p receives the next heartbeat of now-member q at $t_1 + \tau_{hb}$, the p installs a view including q (line 27) at $t_1 + \tau_{hb} < \text{now} + \tau_{hb} \leq \text{now} + \tau_{vc}$ because $t_1 < \text{now}$ and $\tau_{hb} \leq \tau_{vc}$. □ (iii) Let process p be a member with neighbor q in its view, that is not a member. Then, the last heartbeat from q received by p happened before at a time $t_1 > \text{now} - \tau_{hb}$. Otherwise, q would not be in p's list of neighbors. Also, q was a member at t_1, because it was included in p's view then. At $t_2 := t_1 + \tau_{hb}$, the next heartbeat of q is scheduled. If member p receives the heartbeat at t_2, then p removes q from its view (line 31.) If member p does not receive the heartbeat at t_2, then otherTimers(q) expires and p removes q from its view (line 16) as well. If p leaves the group before t_2, then it installs an empty view (line 8.) In all three cases $q \notin view(p)$ at t_2. However, if q becomes a member at t_2 again, then q remains in p's list of neighbors and view, if member p receives the heartbeat. Finally, t_2 happens within the next τ_{vc} time steps, because $t_2 = t_1 + \tau_{hb} < \text{now} + \tau_{hb} \leq \text{now} + \tau_{vc}$. □ (iv) Let process p be a member and there exists another process $q \in view(p)$ and $q \notin list(p)$. This case can only happen between lines 13 and 15. Then, p immediately removes q from its view in line 16. □

View Justification: The view changes in lines 6 and 8 are happening upon join() and leave() events. The view change in line 19 happens when otherTimers(q) at a member process p expires. There, in line 15, $q \in view(p)$ right before and $q \notin list(p)$ (line 12.) Hence, there exists a $q \in view(p) \setminus (list(p) \cup \{p\})$. The view change in line 27 happens after a member process p performs receive(q,TRUE). Then, q is a member but $q \notin view(p)$ (line 26) and $q \in list(p)$ (line 22) before. Hence, there exists a $q \in list(p) \setminus view(p)$ that is a member. The view change in line 31 happens after a member process p performs receive(q,FALSE). Then, q is not a member but $q \in view(p)$ (line 30) and $q \in list(p)$ (line 22) before. Hence, there exists a $q \in list(p) \cap view(p)$ that is not a member. □

7. Applying LGMS

We incorporated LGMS into an application of mobile ad hoc networking to inter-vehicle communication for traffic jam detection on highways [5]. In inter-vehicle communication, vehicles are equipped with computer controlled radio modems allowing them to contact other equipped vehicles in their vicinity. To detect the current size and position of traffic jams, vehicles at the border of the traffic jam send a message to the other end of the congestion. Using LGMS, all vehicles decide on their own if they are at the border of the jam in the following manner.

Slow vehicles traveling in one direction form a dynamic group suspecting to be inside a traffic jam. A vehicle get-

ting caught in a traffic jam and driving at a speed below a threshold v_{jam} issues a join request to its LGMS; when escaping a traffic jam and moving faster than v_{free} again, the vehicle leaves the group. The group identity equals the highway number and the driving direction on this highway.

Member vehicles that are aware of the local membership i.e. other slow vehicles nearby, exchange their position data to determine distributedly which vehicle is the foremost or the last one in the traffic congestion. If a member vehicle has no neighbor who is also a member behind it, then it decides to be at the end of the traffic jam. Analogically, a member vehicle having no neighbor and member in front of it, considers itself being at the beginning of the traffic jam. Vehicles classified as being at the border of the traffic jam then send a message with their current position through the mobile ad hoc network to the other end to detect the length and position of the traffic jam.

We simulated the ad hoc network applied to traffic jam detection in a realistic highway scenario. Therein, 200 vehicles on average drive with at most 36 m/s on a 10 km long road. The density of vehicles is high enough to cause a traffic jam by stopping five consecutive vehicles at the beginning of a simulation run. We set $v_{jam} = 40$ km/h and $v_{free} = 70$ km/h. The transmission range of the radio is 600 m. The timing values concerning LGMS are $\tau_{hb} = 1$ s and $\tau_{vc} = 3$ s. We varied the percentage of equipped vehicles on the road which denote those vehicles participating in the ad hoc network. For all deployment rates in the simulation of LGMS, the installed views reached an accuracy well above 90% despite transient communication failures through occasional packet loss.

8. Conclusion and Outlook

We specified a localized group membership service that suits distributed systems with mobile hosts as found in ad hoc networks. The discussion on related work has shown that defining a partitionable group membership service in asynchronous distributed systems is still an open question and an area of active research. Our specification was mainly inspired by [2] although the application to mobile ad hoc networks made changes inevitable. We extended the idea of allowing different views to exists concurrently in distinct network partitions by reducing the membership problem to the local environment of each node. Thus, we address the problems of frequent topology changes and unreliable communication in mobile ad hoc networks.

We implemented a simple algorithm to solve the stated problem and proved its correctness. An example application for such a localized group membership service demonstrates possible areas of deployment in wireless networks. With this work, we hope to bring together the two research fields of distributed and real-time systems and mobile ad hoc networking. Future goals include investigating the theoretical boundaries of agreement protocols in the context of mobile ad hoc networking.

References

[1] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg. On the formal specification of group membership services. Technical Report TR95-1534, Cornell University, Computer Science Department, Aug. 25, 1995.

[2] Ö. Babaoğlu, R. Davoli, and A. Montresor. Group communication in partitionable systems: Specification and algorithms. Technical Report UBLCS-98-1, University of Bologna (Italy). Department of Computer Science., Apr. 1998.

[3] Ö. Babaoğlu, R. Davoli, A. Montresor, and R. Segala. System support for partition-aware network applications. In *18th International Conference on Distributed Computing Systems*, pages 184–191, May 1998.

[4] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, Dec. 1993.

[5] L. Briesemeister. *Group Membership and Communication in Highly Mobile Ad Hoc Networks*. PhD thesis, School of Electrical Engineering and Computer Science, Technical University of Berlin, Germany, Nov. 2001.

[6] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *15th annual ACM Symposium on Principles of Distributed Computing*, pages 322–330. ACM Press, May 1996.

[7] The Ensemble Distributed Communication System. Web page. Department of Computer Science, Cornell University, http://www.cs.cornell.edu/Info/Projects/Ensemble/.

[8] P. D. Ezhilchelvan, R. A. Macêdo, and S. K. Shrivastava. Newtop: A fault-tolerant group communication protocol. In *15th International Conference on Distributed Computing Systems (ICDCS'95)*, pages 296–306, May 1995.

[9] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proceedings of the eighteenth annual ACM symposium on Principles of programming languages*, pages 353–366. ACM Press, 1991.

[10] The Jgroup Project. Web page. Department of Computer Science, University of Bologna, http://www.cs.unibo.it/projects/jgroup/.

[11] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. A client-server oriented algorithm for virtually synchronous group membership in WANs. Technical Report CS1999-0623, University of California, San Diego, Computer Science and Engineering, July 1999.

[12] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[13] A. Montresor. *System Support for Programming Object-Oriented Dependable Applications in Partitionable Systems*. PhD thesis, University of Bologna, Italy, Mar. 2000. Technical Report UBLCS-2000-10.

[14] The Transis Project Home Page. Web page. Computer Science Department, The Hebrew University of Jerusalem, http://www.cs.huji.ac.il/~transis/.