Maximal Group Membership in Ad Hoc Networks

Mamoun Filali, Valérie Issarny, Philippe Mauran, Gérard Padiou, Philippe Quéinnec

Maximal Group Membership in Ad Hoc Networks

Mamoun Filali[1], Valérie Issarny[2], Philippe Mauran[3], Gérard Padiou[3], and Philippe Quéinnec[3]

[1] IRIT-CNRS-Université Paul Sabatier
filali@irit.fr
[2] INRIA-Roquencourt
issarny@inria.fr
[3] IRIT-ENSEEIHT
{mauran, padiou, queinnec}@enseeiht.fr

Abstract. The notion of Group communication has long been introduced as a core service of distributed systems. More recently, this notion appeared with a somewhat different meaning in the field of mobile ad hoc systems. In this context, we study the group membership problem. After specifying the basic safety properties of such groups and a maximality criterion based on cliques, we propose a group membership algorithm. Lastly, with respect to this criterion, we compare our algorithm with two group membership algorithms for ad hoc environments. Moreover, a formal description in TLA+ has been programmed and verified by model-checking for small networks.

1 Introduction

The notion of Group communication has long been introduced as a core service of distributed systems [1]. More recently, this notion appeared with a somewhat different meaning in the field of mobile ad hoc systems. We introduce group communication protocols in the classical setting. Then, we present the features of mobile ad hoc systems that motivate the design of new definitions and protocols for group communication.

Group Communication Protocols. Group communication services have emerged from two domains : asynchronous distributed systems for fault-tolerant purposes [1, 3, 6] and (distributed) multi-agent systems (MAS) for agent coordination purposes [8]. They have been extensively studied from a formal as well as from a practical standpoint, in the field of distributed computing systems [3] in which:

- the number of connected nodes can be sizeable, but (typically) is not huge.
- the connection of a node to the system remains relatively stable (which does not rule out unexpected failures or disconnections).

In this setting, the group communication service appears as a fundamental service, which allows to multicast messages to a set of nodes (or *group members*), in order to build (more easily) fault-tolerant distributed systems, or to manage the consistency of the group members' interactions, at the application level.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 51–58, 2006.

Group communication is based on the definition of *groups*, i.e. sets of nodes. Messages sent to a group are dispatched to each member of the group. Such a group communication layer is particularly relevant and useful for fault tolerance based on the replication of services or data, or for some classes of applications, such as groupware. Its implementation rests on two basic services:

- a *group membership service*, which maintains a list of current group members, or, more precisely, a representation (a *view*) of this list for each group member. The main problem is to ensure the coherence of each member's view. Changes in the group composition, due to members leaving, or failing, or to new members joining, can be taken into account by the notion of primary view, which defines consistency rules for group evolution [4].
- a *reliable multicast service* which delivers messages to each group member. This level allows to schedule message delivery, depending on the application level requirements in terms of cost and consistency.

Group Communication Protocols in Mobile Ad Hoc Systems. Pervasive computing, which has received an increasing attention during the last decade, brings quite a different setting for distributed systems and applications:

- The ability for any node to join (or quit) the system anywhere, at any time, a priori rules out asymetrical (centralized) protocols in the vein of client/server patterns.
- Scalability becomes a key factor, due to the potentially huge and highly transient set of nodes that make up the system.
- Whereas the design of classical group communication protocols relies on point-to-point communication, *local broadcast* is a natural and often appropriate communication pattern in pervasive computing environments.
- In the same way as networks, groups are not a priori defined. They are rather built up "on the fly", in an ad hoc manner, depending on the requirements and availability of interacting sites at a given time, in a given area.
- Lastly, requirements on autonomy and resource consumption lead to favor robustness and locality, to the detriment of determinism and synchronism, which meets up with the stress layed on scalability.

Thus differences appear on the one hand, in the purpose of groups : in classical group communication systems, the main goal is to determine whether sites belong to a given (unique) group, whilst in the setting of pervasive computing systems, the main issue is to build groups (which may be partitioned) out of neighboring nodes, and, on the other hand, in the way groups evolve : in the classical setting, view updates are incremental and no periodic installation of new views occurs "from scratch".

The notion of group has thus been revisited, in the light of these new constraints, by several recent systems. Our proposal comes within this scope. It is based on the analysis of two existing proposals [2, 8], coming from different, but complementary, fields and concerns, as regards the group communication service, namely mobile ad hoc networks, and (embedded) multi-agent systems. Furthermore, we specify the properties of our protocol in the TLA+ formal framework [5], in order to assess and compare this protocol to existing ones.

2 Group Membership Properties in Ad Hoc Networks

In ad hoc networks, nodes and links continuously appear and disappear. In such a context, group members must exhibit a high connectivity to meet robustness and fault tolerance criteria. The most robust groups of processes of a given size are those that correspond to cliques in the underlying interconnection topology of the network.

Fig. 1. Partitions in cliques

A clique of a graph is any complete subgraph [7]. Figure 1 illustrates partitions in cliques of a graph. Several maximality criteria have been defined on cliques. With respect to the group membership problem, the left partition is better than the right one: groups have more members (i.e. the partition has less cliques). However, two cliques cannot be merged to form a bigger clique. We choose this non-extendible property as a maximality criterion.

Cliques can be used to specify the basic properties of groups. Each grouped process must eventually obtain an *installed view* that contains the members of its current clique. Group membership algorithms should aim at computing partitions of maximal cliques to assign a view to each node. A partition insures that each process belongs exactly to one group and (maximal) cliques provide the most robust groups.

A formal statement of these properties is given by the following definitions: we consider a set of nodes *Node*, the vertices of the network graph. This graph is specified as a set of pairs of nodes and installed views are subsets of *Node*.

$$Graph \subseteq Node \times Node \qquad Views \in [Node \rightarrow \text{SUBSET} \quad Node]$$

Communication properties in ad hoc networks lead to consider specific graphs. More precisely, we assume that a node can send messages to itself and that if a node p can communicate with q, then q can communicate with p. Therefore the graphs we consider for ad hoc networks are reflexive and symmetric:

$$AdHocGraph \triangleq Graph \cup \{\langle p, p\rangle : p \in Node\} \cup \{\langle q, p\rangle : \langle p, q\rangle \in Graph\}$$

In the remainder of this section, we assume the installed views are **non empty** sets. Views must verify consistency properties: an installed view of node p always contains p, views are cliques and views are disjoint sets.

View safety

$$\begin{array}{ll} \forall p \in Node: & p \in View[p] \\ \forall p \in Node: & View[p] \times View[p] \subseteq AdHocGraph \\ \forall p, q \in Node: & (View[p] = View[q]) \ \vee \ (View[p] \cap View[q] = \emptyset) \end{array}$$

First, we specify a local minimal requirement for a group membership service. Views with only one member (*singleton* set) should be avoided: for any couple (p, q) of nodes, if their installed views are eventually reduced to a singleton set, these nodes must not be neighbors. In other words, if a node eventually belongs to a singleton set, its neighbors belong to non singleton cliques.

$$\forall p, q \in Node : View[p] = \{p\} \wedge View[q] = \{q\} \Rightarrow \langle p, q \rangle \notin AdHocGraph$$

A stronger property specifies a maximal criterion : a view cannot be extended with an other distinct view.

$$\forall p, q \in Node : (View[p] \times View[q] \subseteq AdHocGraph) \Rightarrow View[p] = View[q]$$

A restricted form of this property implies that a singleton view cannot extend an existing view.

When all nodes have obtained an installed view, a final requirement states that all views are a covering of the graph : $\bigcup_{p \in Node} View[p] = Node$

From their mutually exclusive property, it follows that the set of views $\{View[p] : p \in Node\}$ is a partition of $Node$.

3 Group Membership Algorithm

The group membership algorithm aims at building non extendible cliques. Each node has the same behaviour. Starting from a *singleton* state, a node performs successive steps bounded by a timeout to reach a final *grouped* state. Figure 2 illustrates this sequence of phases. Such a sequence starts according to a classical approach in distributed algorithms, namely, a diffusing computation. At least one node performs a first local broadcast of a message (possibly after a timeout). Other nodes enter the phase when they receive this message and propagate the current phase by broascasting the same message type.

This sequence of phases is performed by a node until it enters the grouped state. A node remains in a group for a while and repeats the group membership protocol. The lifetime of a group is assumed to be much longer than the time required to build a group. This periodic behavior allows to adapt to the dynamic nature of the network.

During the *Discovering* phase, a node acquires the list of its one-hop neighbors. Then, each node broadcasts this list during the *Publishing* phase. When a

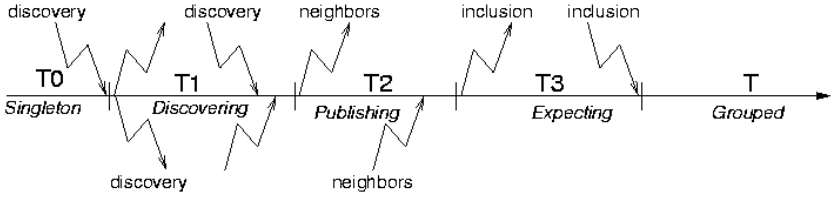

Fig. 2. Phases of the algorithm

node has received all the lists of its neighbors, it has a complete knowledge of its neighborhood at a 2-hops distance. With this knowledge, each node either decides to wait an inclusion request from an other node or to compute a clique and broadcast this view to target members. This decision relies upon a total priority order defined on nodes. A node evaluates a new view if and only if its priority is greater than all its 2-hops neighbors.

Figure 3 illustrates a node with its neighbors and 2-hops neighborhood. Node priority is assumed to be equal to their identity. This node of maximal priority will decide to build a view including either the nodes {8,10,15} or the nodes {7,11,15}. The clique {6,15} could also be considered, but larger cliques should be preferred.

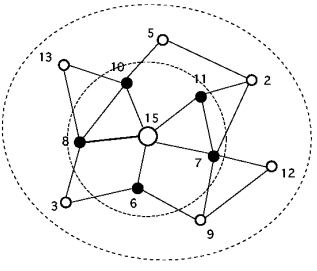

Fig. 3. Neighborhoods

The main idea for choosing the maximum over the 2-hops neighborhood is that the same node cannot be twice chosen to be a member of two distinct views. If the same node could be selected by two other nodes, then the distance between these two nodes should be at most 2. It follows that the node will be selected by at most one of them since the priority of a node that selects a view is greater than the priority of any other node at distance 2.

Properties of such algorithms can only be specified under stability assumptions on the underlying network during a bounded period. Henceforth, we assume that the underlying network connections are stable from the beginning of the protocol until all nodes are grouped. However, if this stability condition does not hold, the algorithm still guarantees the three view safety properties (See section 2), but cannot guarantee any longer the maximality property (non extendible property).

3.1 State Transition Graph

Figure 4 describes the state transitions of a node. A transition is performed when a message is received or a timeout occurs. During a phase, specific message type(s) can be broadcast (`!m`) and/or received (`?m`).

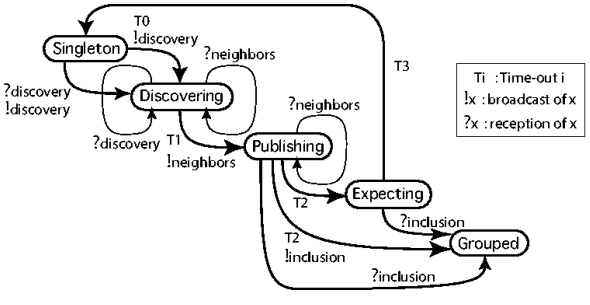

Fig. 4. State transition graph

There are three message types containing the sender's identity: *discovery*, *neighbors* and *inclusion*. A *neighbors* message also contains its sender neighbors and an *inclusion* message contains the resulting view.

Each node maintains the following local variables: its current state, a set of neighbors, the neighborhood of each of its neighbors and a view defined when the node is grouped.

3.2 Message Handling

For a singleton node, the reception of a first *discovery* message or a timeout `T0` occurrence triggers a transition toward the *Discovering* state and a broadcast of a `discovery` message toward reachable nodes. The sending node is recorded as a new neighbor and the current state remains or becomes *Discovering*. The discovery process is propagated by broadcasting a `discovery` message. If the node is already in the *Discovering* state, the same actions are performed but the state remains unchanged.

In the *Discovering* or *Publishing* state, if a node receives a `neighbors` message, its neighborhood is updated with the content of the message, namely the neighbors of the sending node.

In the *Expecting* or *Publishing* state, if a node receives an `inclusion` message, it accepts the content of the message as its current view and becomes grouped.

3.3 Timeout Handling

When a timeout occurs, according to the current state of the node, a transition is performed. We assume the propagation time of a message to be the dominant duration. Timeout choice rests upon the following constraints:

- Timeout `T0`: must be long enough to mask the asynchronous timing of phases among nodes ; nodes acquire a weak synchronous behavior after the *Discovering* phase and remain at most shifted from one phase.
- Timeout `T1`: in the *Discovering* state, a node has to broadcast its identity and receive its neighbors. As neighbors broadcast their identity as soon as they receive a `discovery` message, `T1` must be at least longer than 2 broadcasts (plus enough time for local processing) ;
- Timeout `T2`: same constraints as `T1` in so far as the nodes have the same behavior with respect to the `neighbors` messages.
- Timeout `T3`: in the *Expecting* state, a node has at most to wait for the choice of a view and its broadcast. Therefore, `T3` must be longer than this duration.

From the *Singleton* state, a timeout occurrence triggers a transition toward the *Discovering* state and a local broadcast of a `discovery` message that contains the sending node name.

From the *Discovering* state, a timeout occurrence triggers a transition toward the *Publishing* state and a local broadcast of a `neighbors` message which contains the current neighbors list of the sending node.

From the *Publishing* state, a timeout occurrence either leads to evaluate a new view if the current node has the maximal priority over its 2-hops neighborhood and to enter the *Grouped* state or to wait for an `inclusion` message in the *Expecting* state.

From the *Expecting* state, when a timeout occurs, the node returns into the *Singleton* state.

Number of Messages. In the best case, the network itself is a clique, and one iteration of the protocol is enough to build a group with all the nodes; if N is the number of nodes, it needs N broadcasts (`discovery` message) + N broadcasts (`neighbors` message) + 1 broadcast (`inclusion` message).

In the worst case, the network is linear and nodes are placed according to their priority (that is, nodes 1 and 2 are connected, nodes 2 and 3 are connected,...). Then, each iteration builds only one group with the two highest priority nodes. Then $N/2$ iterations and $O(N^2)$ broadcasts are required.

4 Related Algorithms

With respect to the general group membership problem, our study is mainly concerned by group construction algorithms in partitionable networks [9]. In this section, we first make some general remarks about group construction algorithms used in partitionable networks, then, we present the main features of two algorithms: the first one is used in the context of ad hoc networks [2], while the second is used in the context of multi-agent systems [8].

Views are close to the underlying network topology. A view is a connected component: it contains processes that can be reached (through one or several hops) from each other. However, this definition is usually strengthened: a view contains processes that can reach each other in one hop, i.e., a view is a clique of the underlying network. Moreover, initially, either a node knows its immediate neighbors or has to discover them through a broadcasting primitive.

The algorithm of [2], the starting point of our study, concerns group management in mobile ad hoc networks. Although, the algorithm is also concerned by security aspects as well as by application level aspects, we discuss here group membership aspects only. First, in order to take into account the dynamic topology of ad hoc networks and the resource consumption constraints of mobile applications group maintenance is periodic. Periodically, during a discovery phase, a node builds dynamically its neighbors set. Once a node knows its neighbours[1], it sends to *one* of them (the maximal one) its set of neighbours. Then, nodes which have received some sets of neighbors may choose to build a connected view. This algorithm cannot avoid extendible views. However, it has good properties with respect to the number of messages and the minimization of energy.

The algorithm of [8] concerns also group membership. This algorithm is based upon the construction of a common knowledge amongst neighboring nodes. Each node broadcasts the list of its neighbors which is assumed to be initially known.

[1] Timeouts are used for bounding the discovery period.

Then, each node gathers the neighborhood of its neighbors. Once this common knowledge is built, the following stages differ from our algorithm, as this knowledge is used to define views through a consensus: in the last stage of a loop iteration, the participants have to agree on a common view. In order to avoid cycles, clique choices have to be restricted over iterations and the algorithm tolerates extendible cliques to ensure the convergence of the loop. This algorithm is more concerned by the complexity issues for computing cliques: basically, it uses pre-specified thresholds over the size of cliques.

5 Conclusion

In this paper, we have been mainly concerned by group membership protocols in ad hoc networks. After specifying its basic safety properties and a maximality criterion about the installed views, we have proposed a group membership algorithm. Lastly, with respect to this criterion, we have compared our algorithm with two group membership algorithms. We have also specified the properties of the algorithm as well as its description in the TLA+ formalism. Currently, we have performed model-checking experiments with the TLC tool [5]. On small size graphs (6 nodes), we have been able to automatically check the correctness of our algorithm. We are now working on its formal correctness (any number of nodes) through theorem proving techniques.

References

1. K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993.
2. M. Boulkenafed, V. Issarny, and J. Liu. Group management for in-home ad hoc networks. In *ECRTS International Workshop on Real-Time for Multimedia - Special Focus on Real-time Middleware for Consumer Electronics (RTMM)*, June 2004.
3. G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications : A comprehensive study. *ACM Computing Surveys*, 33(4):427–469, December 2001.
4. R. De Prisco, A. Fekete, N. Lynch, and A. Shvartsman. A dynamic view-oriented group communication service. In *Proceedings of the 17th annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 227–236, June 1998.
5. L. Lamport. *Specifying Systems : The TLA+ language and tools for Hardware and Software Engineers*. Addison Wesley Professional, 2002.
6. S. Mishra, C. Fetzer, and F. Cristian. The Timewheel group communication system. *IEEE Transactions on Computers*, 51(8):883–899, August 2002.
7. S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, pages 215,217–218. Addison-Wesley, Reading MA, 1990.
8. P. Tosic and G. Agha. Maximal clique based distributed group formation for task allocation in large-scale multi-agent systems. In *Proceedings of the International Workshop on Massively Multi-Agent Systems, Kyoto, Japan*, December 10-11 2004.
9. Özapl Babaoğlu, R. Davoli, and A. Montresor. Group communication in partitionable systems : Specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4):308–336, April 2001.