A Peer-to-Peer Membership Notification Service*

Roberto Baldoni and Sara Tucci Piergiovanni

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Rome, Italy {baldoni,tucci}@dis.uniroma1.it

Abstract. The problem of providing a peer with a good approximation of the current group membership in a peer-to-peer (p2p) setting is a key factor to the successful usage of any application-level multicast protocol (e.g. gossip based protocols). A p2p setting makes this problem hard to be solved due to the its inherent dynamic and asynchronous nature. This paper studies the problem of implementing a fully distributed, also called p2p, Membership Notification Service (MNS) which is able to handle any number of simultaneous join and leave while allowing reliable delivering of messages among peers which remain permanently alive inside the group.

1 Introduction

We are interested in studying group membership in very large scale peer-to-peer environments formed by processes sharing a common interest. In this setting, hundreds of thousands of processes communicate through application-level multicast protocols over an overlay network formed by the peers themselves [10, 4]. This environment is inherently asynchronous and dynamic because peers continuously join and leave the system. This implies that the multicast protocol, to be effective, has to rely on a group membership service [1] to individuate at each point in time the set of intended peer receivers for each multicast message.

Group membership has been extensively studied in the literature in the context of group communications. Traditionally, group membership [5] supports process group communication [2] with the following two objectives [11]: (i) determining the set of processes that are currently up and (ii) ensuring that processes agree on the successive values of this set. These group membership approaches require "long enough" periods of time in which (i) no membership changes occur and (ii) the underlying system model shows a synchronous behavior [6]. The scale and dynamic nature of a p2p environment make the requirement of a "long enough" period of stability and synchrony problematic to discharge in practice.

* The work described in this paper was partially supported by the Italian Ministry of Education, University, and Research (MIUR) under the IS-MANET project.

[1] The multicast can also directly embed the membership management, but in the following we maintain separated these two concerns: multicast communication of application information and group membership management.

Recently, in the context of WAN, Anker et al. [1] proposed the notion of *Membership Notification Service* (MNS) which provides each process with an approximation of the current group membership, without being synchronized with the message stream. This approach allows handling any number of simultaneous join/leave events concurrently and allows message reliability among those members that remain permanently alive against on-going membership changes. The authors proposed a client/server implementation of a MNS. More specifically, there is a set of servers and each server is in charge of (i) being the access points for joining nodes, (ii) tracking the departures of processes (both failures and voluntarily leaves) and (iii) providing views of the membership to whom requested them.

The aim of this paper is to study what are problems which arise in implementing a fully distributed (or p2p) MNS in a p2p setting and, then, to propose a solution.

The contribution of the paper is twofold, the paper firstly presents two impossibilities results that delimitate under which assumptions a p2p MSN implementation can be realized. Secondly, it introduces a p2p MNS solution that manages concurrent leaves. The presented solution dynamically builds and maintains an overlay topology in which processes are partially ordered by a *rank* which is assigned to a process at join time. This weak order is taken into account at leave time. The algorithm serializes any two concurrent leave operations executed by neighbor processes in the overlay topology that could lead to a partition of the topology itself. These departures are ordered by process rank. All the other concurrent leave operations are not serialized.

The paper is organized as follows. Section 2 presents the system model including the group membership management. Section 3 formally introduces the MNS specification, the impossibility results and the circularity problem. Section 4 shows a p2p MNS implementation.

2 System Model

The system consists of an unbounded but finite set of processes Π[2]. Any process may fail by crashing. A process that never fails is correct. The system is asynchronous: there is no global clock and there is no timing assumption on process scheduling and message transfer delays. Each pair of processes p_i, p_j may communicate along point-to-point reliable links. To simplify the description without losing generality, we assume the existence of a fictional global clock, whose output is the set of positive integers denoted by $\mathcal{T}$.

Group Membership. Each process $p_i \in \Pi$ may become member of a group G. Once member, it may decide to leave the group. To this aim p_i may invoke the following operations: `join(G)` to enter G and `leave(G)` to exit the group. The set of processes constituting the group G at time $t \in \mathcal{T}$ is denoted as $G(t)$.

[2] Note that in the informal parts of the paper we use the term "peer" as a synonym of "process".

At any time t, $G(t)$ is a subset of Π with size unbounded but finite. The rules defining the membership of G are the following:
1. a process $p \in \Pi$ becomes a member of G immediately after the completion of `join(G)`.
2. a process p ceases to be member of G immediately after the completion of `leave(G)`.
3. a process p may become member of G at most once [3].

A group member $p \in G$ is *stationary* if it is correct and never invokes `leave(G)`. A group member $p \in G$ is *transient* if it is correct and eventually executes `leave(G)`.

Note that, since $G(t)$ is unbounded and finite at any point of time, the number of stationary processes is unbounded and finite, as well.

Membership Management. To abstract in a general manner the membership management we consider that each process can locally access a distributed oracle. Each process p_i invokes the `join(G)` and `leave(G)` operations through the *MNS local module* MNS_i that is in charge of the actual execution of these operations. Each MNS_i module provides p_i with a *local view* of the group, i.e. a *list of processes* representing the current membership of the group as perceived by MNS_i. We assume that MNS_i crashes only when p_i crashes.

Upon the invocation of `join(G)` by p_i, MNS_i generates an event denoted as inv_i(`join(G)`), then MNS_i is granted permission to access the group on behalf of p_i. After that, MNS_i returns to p_i with an upcall by generating the event res_i(`join(G)`), thus at this point $p_i \in G$.

Upon the invocation of `leave(G)` by p_i, denoted as inv_i(`leave(G)`), MNS_i obtains permission for p_i to leave the group. After that, MNS_i returns to p_i with an upcall res_i(`leave(G)`), thus from this time on p_i is no longer a member of G.

Note that MNS_i may provide p_i with a local view even when p_i is not a group member. In this case we call the view $access\_view_i$. As long as p_i belongs to the group, the local view is called $group\_view_i$. Only between events res_i(`join(G)`) and res_i(`leave(G)`) we say that $p_i \in group\_view_i$.

3 MNS specification

The view information for one group can be represented as a knows-about directed graph $K = (\Pi, E)$ [9]. For each pair of processes p_i, p_j, there will be an edge (p_i, p_j) in the graph if $p_j \in group\_view_i$, and an edge (p_j, p_i) if $p_i \in group\_view_j$. There exists an edge (p_i, p_i) for every process p_i such that $p_i \in group\_view_i$ [4]. This graph actually represents the overlay network to be used as underlying communication network by an application-level multicast protocol in a peer-to-peer environment.

[3] This is not a restriction because the process may join with another identifier.
[4] Note that there exists an edge (p_i, p_i) even for each faulty member p_i that crashes before generating res_i(`leave(G)`).

3.1 Specification

Safety. Since view information is propagated along edges of the knows-about graph, once joins and leaves cease, every stationary member p_i belonging to the graph should have for each stationary member at least one path formed by stationary members [5]. This is necessary because even though leaves and joins no longer occur, crashes are still possible. Such crashes could partition the set of stationary members. Therefore, if this condition is satisfied, the view of each stationary member eventually includes all stationary members. Formally,

Property 1 (Safety). Let $K = (\Pi, E)$ denote the knows-about graph at time t s.t. no edge (p_i, p_i) will be added or removed for each $t' > t$ (i.e., joins and leave cease at time t). Let us consider the subgraph $K_s = (S, E_s)$ such that
(i) $p_i \in \Pi$ and p_i is stationary $\Leftrightarrow p_i \in S$
(ii) $\forall\ p_i, p_j \in S,\ (p_i, p_j) \in E \Leftrightarrow (p_i, p_j) \in E_s$.
Then, $\forall p_i, p_j \in S$ there exists an edge (p_i, p_j) in the transitive closure of E_s for each $t' > t$.

Liveness. A trivial group membership implementation may maintain *safety* by blocking the completion of each `join(G)/leave(G)`.

Then, to avoid *static* implementations the following property holds:

Property 2 (Liveness). The execution of the *`join(G)` and `leave(G)`* operations requires *finite* time.

3.2 Impossibility results

The following impossibility results stem from the general assumption that $access\_view_i$ is a random set of processes belonging to Π, without any relation with the current membership. Unfortunately, as we see later, to guarantee the MNS specification even $access\_view_i$ has to satisfy some property (stated in Corollary 1). While this property is very lightweight, it nonetheless necessarily introduces a circularity problem.

Impossibility Result 1 *If there exists a time $t \in \mathcal{T}$ s.t. $G(t) \equiv \emptyset$, the MNS specification cannot be guaranteed.*

Proof. (sketch) Let us suppose by contradiction that at some point of time t, $|G(t)| \equiv \emptyset$.

Assume a process p_i executing `join(G)` produces the inv_i(`join(G)`) event while $|G(t)| \equiv \emptyset$. p_i does not know whether the group is empty or not as $access\_view$ is neither complete nor accurate. p_i can send a JOIN message to its $access\_view$ but cannot get any acknowledgement (like any concurrent joining process) since $G(t)$ is empty. To respect Liveness, p_i has become a member after

[5] Our Safety specification is partially inspired by the group membership specification in [9].

a finite amount of time exploiting a time-out strategy [6]. Then, at time $t+T$ p_i concludes to be alone in G and includes in $group\_view_i$ only itself. Because of the asynchrony of the underlying system, another process p_j with $p_j \notin access\_view_i$ and $p_i \notin access\_view_j$ can decide to join. As p_j does not "see" p_i, it uses the same strategy and generates $res_i(\texttt{join(G)})$ including in $group\_view_j$ only itself at time $t+T$. If both p_i and p_j are stationary no edge connects them at time $t' \geq t+T$. If no other join and leave occur there is no way to add that edge at a later moment. Hence, no edge will connect them for each $t' \geq t+T$ violating Safety.

Lemma 1. *Let us suppose that $|G|$ is never empty. Then, any process p_i cannot generate $res_i(\texttt{join(G)})$ until there exist at least one edge $(p_i, p_j) \in E$ and one edge $(p_j, p_i) \in E$.*

Proof. (sketch) Let us suppose that $res_i(\texttt{join(G)})$ is generated at time t and that $G(t)$ contains a stationary member p. By the way of contradiction, let us suppose that does not exist any edge (p_i, p_j) in E at time t. However, after $res_i(\texttt{join(G)})$ p_i has an edge $(p_i, p_i) \in E$ at time t. If p_i is also stationary then $G(t)$ contains two stationary processes and no edge in the transitive closure of E. If no other join and leave occur there is no way to add that edge in a successive moment. No edge will connect them for each $t' \geq t+T$, violating Safety.

Impossibility Result 2 *If there exists a time $t \in \mathcal{T}$ s.t. $G(t)$ contains no stationary member, the MNS specification cannot be guaranteed.*

Proof. (sketch) Let us suppose by contradiction that there exists a point of time $t \in \mathcal{T}$ s.t. $G(t)$ does not contain stationary members. From Lemma 1 every joining process has to establish two edges with a process p_j, before generating $res_i(\texttt{join(G)})$. From Liveness it has to establish those edges in a finite time. Without loss of generality suppose that at time t, $G(t)$ comprises k faulty members and c transient processes. Taken any subset $S(t) \subseteq G(t)$, of one process p_j, that process is either faulty or transient. Let us assume that:

1. p_j is transient and belongs to G between times t_j^J and t_j^L.
2. p_i generates $inv_i(\texttt{join(G)})$ and sends at time t a $JOIN$ message to each member of $G(t)$.

As the system is asynchronous, the delay experienced by $JOIN$ on the fair lossy link connecting p_i to p_j, could be greater than $t_j^L - t_j^J$ and then the message of p_i would not reach p_j. Moreover, p_j does not yet know p_i, so p_j cannot communicate with p_i before the $JOIN$ arrives to p_j. Then, p_i cannot establish any edge with p_j. p_i cannot establish in a finite time any edge unless some other stationary member will join the group. However, no stationary member can surely join in a finite time for the same reason that blocks p_i. Thus, p_i waits for an infinite time violating Liveness.

[6] The strategy can encompass mechanisms such as setting a timeout T or retransmitting the JOIN message k times.

From Impossibility Result 2, the following Corollary holds:

Corollary 1. *If $access\_view_i$ does not eventually contain at least one stationary member, the MNS specification cannot be guaranteed.*

This constraint on *access_view* poses a *circularity problem* when the MNS is implemented in a pure p2p fashion, i.e. it is implemented in a fully decentralized manner by members themselves and no process plays a special role from the beginning. In this case, to fill the *access_view* in order to be compliant with Corollary 1, a run time discovery has to be performed. This discovery cannot be push-based (from the current members of the group to the newcomer): none can indeed provide the newcomer with a view as no member of the group knows the newcomer [7]. Thus, to discover a current peer, the newcomer has to contact someone (e.g. a special process) that knows some peer. Following a pure peer-to-peer approach (where there is no special process), only a peer may have this knowledge. Then, to know a peer, the newcomer must already know a peer: a classic instance of the hen-and-egg problem.

Circularity, in these systems, may be avoided by assuming either that eventually the newcomer will somehow know someone inside the group or the existence of special processes constantly known by all other processes from the beginning—at the cost, however, of losing a pure peer-to-peer approach.

4 A p2p MNS Implementation

In this section we provide a p2p MNS implementation. In particular, the MNS is implemented by the peers themselves where each peer only has only a partial view of the group[8, 7]. The interested readers are referred to [3] for a performance analysis of the algorithm and its comparison with [8].

The proposed algorithm may concurrently handle join/leave operations generating, in a decentralized manner, knows-about graphs respecting Safety. The resulting graphs show a particular structure in which each member has around itself a clique of at least $f + 1$ members, where f is the number of tolerated failures. The other important feature of the algorithm consists in imposing a partial order on processes to manage concurrent leaves that may partition the graph. The algorithm also exploits heartbeat messages to monitor node failures.
Data Structures. The variable $group\_view_i$ is the union of two different variables: $sponsors_i$ and $sponsored_i$. $sponsors_i$ is a list of processes (identifiers) which guarantee to p_i the connection[8] to the group, i.e. upon the join operation the list contains all processes the grant p_i the permission to enter the group, then if some of these sponsors leaves the list will contain some other process that replaces the left one. $sponsored_i$ is a list of processes (identifiers) which p_i is responsible for in terms of connection. A variable $rank_i$ gives an indication

[7] The number of potential newcomers is unbounded. As a consequence the identifiers of potential newcomers cannot be available at design time.

[8] The connection is intended here as the connection to the overlay in terms of knows-about relation.

of the position of p_i in the graph, inducing a partial order on nodes. A boolean variable *leaving* is initialized to $\perp$.
Initialization of the group. A set of processes $\{p_1, ...p_{f+1}\} \subseteq \Pi$ totally interconnected and defined in the initialization phase instantiates the group[9]. All these processes have rank $rank_i = 0$. They are special processes, they never leave the group.

Join Management. Rules of the algorithm:

- MNS_i sends a JOIN message to $access\_view_i$ [10]
- When MNS_i receives a JOIN message from MNS_j and $p_i \in group\_view_i$: (1) MNS_i inserts p_j in $sponsored_i$; (2) it sends an acknowledgement to p_j along with its own rank $rank_i$.
- When MNS_i receives $f+1$ acknowledgments: (1) MNS_i includes in $sponsors_i$ all the senders and p_i; (2) it sets $rank_i = max(rank_k, \forall senderp_k)+1$ and (3) returns to p_i generating $res_i(\texttt{join(G)})$. From this time on with an heartbeat mechanism all $sponsors_i$ are monitored. Each time a sponsor is suspected to be faulty, MNS_i tries to re-establish the missed connection searching another sponsor p_j with $rank_j < rank_i$.

Leave Management. Rules of the algorithm:

- MNS_i (i) sets $leaving_i = \top$ and (ii) sends a LEAVE message to $sponsors_i$, so composed $\langle LEAVE, sponsored_i, rank_i \rangle$;
- When MNS_i receives a LEAVE message $\langle LEAVE, sponsored_r, rank_r \rangle$ from MNS_j and $rank_j > rank_i$ and $leaving_i = \perp$: (1) MNS_i inserts $sponsored_r$ in $sponsored_i$; (2) it sends an acknowledgment to p_j and (3) sends a message $\langle NEWSPONSOR, oldsponsor = p_j \rangle$ to $sponsored_r$.
- When MNS_i receives an acknowledgment from its sponsors: (1) discards p_i from $sponsors_i$ and (2) returns to p_i generating $res_i(LEAVE(G))$.
- When MNS_i receives $\langle NEWSPONSOR, oldsponsor_r \rangle$ from MNS_j and $oldsponsor_r \in sponsors_i$: MNS_i includes p_j in $sponsors_i$ and discards $oldsponsor_r$ from $sponsors_i$.

Thanks to ranks, it is possible to induce a partial order on the nodes. In practice, when two nodes p_i, p_j with rank $rank_i < rank_j$ want to concurrently leave, a partition may occur if they actually leave at the same time. The algorithm sequences the leaves, by allowing a leave of a process of rank $rank_j$ only if none of its sponsor p_i with rank $rank_i < rank_j$ is concurrently leaving. Note that p_j remains blocked as long as new sponsors of p_j are concurrently leaving. Eventually, if all processes with rank lower than $rank_j$ leave, then p_j will have as sponsors processes with rank 0. Since by construction these processes never leave (they are stationary), then also p_j eventually will leave (Liveness).

[9] Impossibility results are circumvented because of the presence of these processes.
[10] The mechanism to fulfill *access_view*, addressing Corollary 1, will be discussed in the reminder of this Section.

Each process, if no failures occur, maintains at any time a knows-about graph with connectivity at least equal to $f + 1$. If some failure occurs during overlay changes, a recovery mechanism restore the connectivity of graph. If overlay changes (joins/leaves) subside, the resulting knows-about graph has connectivity $f + 1$ and remains always connected until f failures occur. Safety is maintained until these f failures occur. Anyway, a restoring mechanism will restore connectivity until only stationary processes are in the overlay. From this time on connectivity among stationary members is always guaranteed.

References

1. Tal Anker, Danny Dolev and Ilya Shnayderman: Ad Hoc Membership for Scalable Applications. Proceedings of 16th International Symposium on DIStributed Computing, (2002)
2. Kenneth Birman and Robert van Renesse: Reliable Distributed Computing with the Isis Toolkit. IEEE Computer Society Press (1994)
3. Roberto Baldoni, Adnan Noor Mian, Sirio Scipioni and Sara Tucci Piergiovanni: Churn Resilience of Peer-to-Peer Group Membership: a Performance Analysis. International Workshop on Ditributed Computing (2005), to appear.
4. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Antony Rowstron: Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure. IEEE Journal on Selected Areas in communications (2002)
5. Gregory Chockler, Idit Keidar, Roman Vitenberg: Group Communication Specifications: a Comprehensive Study. ACM Computing Surveys 33(4): 427-469 (2001)
6. Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernardette Charron-Bost. On the Impossibility of Group Membership. In 15th Annual ACM Symposium on Principles of Distributed Computing, (1996)
7. Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Petr Kouznetsov, Anne-Marie Kermarrec: Lightweight Probabilistic Broadcast. ACM Transactions on Computuer Systems 21(4): 341-374 (2003)
8. Ayalvadi J. Ganesh, Anne-Marie Kermarrec, Laurent Massoulié: Peer-to-Peer Membership Management for Gossip-Based Protocols. IEEE Transactions on Computers 52(2): 139-149 (2003)
9. Richard A. Golding and Kim Taylor: Group Membership in the Epidemic Style. Technical Report UCSC-CRL-92-13, University of California, Santa Cruz (1992).
10. John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, James W. O'Toole: Overcast: Reliable Multicasting with an Overlay Network. Proceedings of the 4th Symposium on Operating System Design and Implementation, San Diego (2000)
11. André Schiper and Sam Toueg: From Set Membership to Group Membership: A Separation of Concerns, Technical Report, EPFL, Lausanne, (2003)