Ad Hoc Membership for Scalable Applications

Tal Anker[1,2], Danny Dolev[2], and Ilya Shnayderman[2]

[1] Radlan Computer Communications, Israel
[2] School of Engineering and Computer Science., The Hebrew University of Jerusalem, Israel {anker,dolev,ilia}@cs.huji.ac.il

Abstract. The paper describes an ad hoc approach realized in a practical distributed transport layer called Xpand [1] to improve the message transmission service over a WAN. The current technology focuses on applications that require strong semantics. The ad hoc membership approach increases the asynchrony of handling both control and message flows in order to overcome membership changes with minimal effect on ongoing streams of messages. This approach is beneficial for a variety of applications. Its implementation is expandable to address stronger semantics for applications that need them.

1 Introduction

There are two major widely-recognized approaches toward building distributed data-based applications and replicating objects. The first approach, known as Group Communication Systems (GCSs) [2], presents powerful building blocks for supporting consistency and fault-tolerance in distributed applications, whereas the Paxos [3] approach focuses on ordering actions among a group of servers. While implementing either of the approaches, one observes the system's performance to degrade significantly as group size and message transmission volume increase. These performance problems become even worse in wide-area network environments where message latency is often high and unpredictable. Only few implementations specifically address WAN environment, e.g., Spread [4], InterGroup [5] and *Xpand* [1]).

In this paper we present an ad hoc membership algorithm that is implemented in *Xpand* [1]. The membership services offered by Xpand are designed to be flexible. On one hand, they can be used for maintaining a group of participants and potential participants of a group by applications that do not need strong semantics.[3]. On the other hand, while focusing on efficiency, these services allow to consistently provide stronger semantics for applications that require it.

The efficiency of the ad hoc membership is a result of the separation it implies between message flow, membership algorithms, and failure detection (Network Event Notification Service). Moreover, two separate reliable message dissemination services for control messages and for application messages are used. This

[3] For instance, these applications may not want to block or perform roll back actions per change in the system as GCS or Paxos require.

enables us to remove reliability maintenance overhead from the critical path of delay-sensitive applications.

Within the ad hoc approach, the membership notifications serve only as *approximations* of the current group membership, without being synchronized with message stream. Continuous message reliability is guaranteed only among those group members that remain permanently alive and interconnected. This approach allows handling any number of simultaneous join/leave events concurrently, without waiting for the network or the group of members to stabilize. The traditional membership approaches require a stable period of time in order to consent on a new membership. As a result, they tend to limit the adoption of joining processes during periods of instability. The ad hoc approach provides stability of message flow against on-going membership changes, which is better fit for large groups and for wide area networks where the probability of unstable periods increases. The scalability of Xpand is accounted for by both its hierarchical architecture and the ad hoc membership approach.

The development of the ad hoc membership approach faces three challenges, interrelated to one another. (1) Architectural issue: which channel to prefer for which message: the more reliable "sequential" channel (a slow one), or the less reliable concurrent channel where a loss of some messages does not slow down the delivery of other messages. (2) Algorithmic issue: the dual channel architecture increases the asynchrony among various blocks that provide membership service and, as a result, produce conflicting race conditions that need to be controlled. (3) Design issue: how to design a system that takes advantage of the above, while still maintaining the ability to provide a stronger semantics for applications that require it.

The ad hoc approach best suits the requirements of collaborative networking applications, fault-tolerant applications that require weaker synchronization among a set of servers, one to many push applications (e.g., stock market monitoring) and the like. In Section 2.1 we present more examples of such applications.

One can try to obtain the properties of an ad hoc membership using other WAN toolkits, like combining Spread [4] and PBcast [6], though the challenge remains of providing the properties we look for in an efficient and robust way. Moreover, Spread and similar toolkits assume a predefined global set of servers, whereas Xpand [1] does not require an a priory knowledge of the potential set of group members.

The ad hoc membership is fully implemented within the Xpand middleware. Section 5 presents performance results of the implementation showing that membership changes have a negligible effect on the existing message flows.

2 Xpand Membership Service Model

Xpand membership algorithm is optimized to fulfill the following properties:

Smooth-Join: A new member joins the group without affecting the current message flow in the group;

Fast-Join: Once a joiner is registered in its LAN, it can start emitting messages to the group;

Dynamic Membership: Allows processes to dynamically join and leave the group in a WAN setting.

Xpand membership algorithm achieves these three properties without compromising the following basic message delivery services of Xpand:

FIFO order: Any receiver that starts receiving messages from a given source will get all emitted messages in FIFO order from the moment it joins the message flow;

Gap-Free: Two processes that remain connected during consecutive membership changes continue to receive each other's messages without any gap in the message stream.

The above-listed membership services combined with Xpand's reliable multicast service can be used as a dynamic and reliable point-to-multipoint service layer, on top of which stronger semantic services can be built. For instance, a virtual synchrony layer can be implemented as an extension of the ad hoc membership service Moshe [7]. Another example is Paxos, which can be implemented on top of our ad hoc service by having the leader communicating to existing majorities via the ad hoc services to carry out the three phases of Paxos [3].

2.1 Implementation Issues

In order to enable maintaining of large groups of clients, Xpand is implemented using a two-level hierarchical architecture. Each LAN is represented by a delegate that coordinates the protocol activities of this LAN within the WAN group.

In the current implementation, we assume that the number of LANs with processes that emit messages to the group is of the order of several dozens. The number of processes per LAN is assumed to be of the order of a couple of hundreds. These limitation is imposed due to the need for maintaining state information per sender and per receiver. We can further improve the implementation by enabling a client to join either as a receiver-only, a sender-only, or as a full member (i.e. both a sender and a receiver). Such an approach will enable to increase the number of processes, while keeping a reasonable amount of state information. To increase scalability, the architecture can be used for aggregation. A delegate can represent a collection of senders in its LAN, and the senders should not be explicit members of the WAN group. They will forward their messages to the delegate which will emit them to the WAN group.

We expect the above improvements to enable the ad hoc system to handle tens of thousands of clients.

The protocol is presented in this paper as a collection of state machines within various processes. The actual implementation of the protocol closely followed the state machines. This method enabled us to easily detect protocol and implementation bugs that appear in unforeseen transitions. We found out that by adding history (log) within every state machine we were able to drastically shorten the debugging time.

The ad hoc approach best suits the requirements of collaborative networking applications where the importance of service timeliness prevails over message reliability and consistency requirements are weaker than those of replicated database. For example, multimedia conferencing and distance learning applications benefit from the ad hoc GCS services in workspace sharing and parties coordination [8, 9].

Another example is a loosely coupled set of servers. For example, in the fault-tolerant video-on-demand service by Anker et al. [10], a GCS based on the ad hoc membership service can be used for video server fault-tolerance and QoS negotiation[4]. The efficiency of handling membership changes by our protocol allows for smooth client hand-offs with smaller video-frame buffers at the client.

Yet another application is a case when multiple servers emit information to a loosely coupled set of clients. Each client wishes to receive the stream as soon as possible. For example, in stock market monitoring application messages are generated at several centers spread all over the world. The accessibility to the information is critical, but there is no justification for blocking the stream of messages during network changes. In an on-going audio conference, a newcomer can benefit from Xpand in the sense that Xpand will deliver a reliable multicast service from every session participant to the newcomer (and vice versa) right after the establishment of the corresponding new connections, without affecting previous established connections.

3 The Environment Model

We assume the message-passing environment to be asynchronous, processes communicate solely by exchanging messages and there is no bound on message delivery time. Processes fail by crashing and may later recover, or may voluntarily choose joining or leaving. Communication links may fail and recover.

Xpand exploits the following underlying services:

- A network event notification service (Subsection 3.1), through which Xpand learns about the status of processes and links;
- An integral reliable FIFO communication layer within the network event notification service. It is assumed that notification events and messages delivered via this service are causally consistent;
- A reliable point-to-point service for control messages of the protocol;
- A simple reliable point-to-multipoint service *in a LAN* for control messages of the protocol.[5]

The reliability of the above transport services means that messages sent via them are eventually received, or the recipient will eventually be suspected by

[4] The effect of not using a virtual synchrony in this VoD implementation is that the client's machine may receive, in transient situations, duplications of video frames.

[5] This mechanism is not a substitute for the general services of Xpand, but a simple mechanism focused on guaranteeing reliability over a small number of messages exchanged among protocol participants in a single LAN.

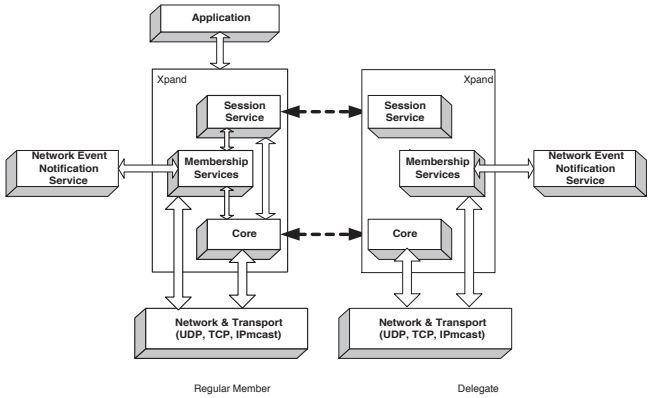


Fig. 1. The Layer Structure of the Protocol Participants.

the network notification service. For efficiency reasons, all the reliable transport services are used only for control messages (i.e. for protocol messages but not for user application messages).

3.1 Network Event Notification Service (NS)

Clients use the notification service to request joining or leaving the groups, or to get updated information regarding the group. The notification service accumulates and disseminates failure detection information along with information about these requests. The service is provided to clients by an interface that consists of the following basic functions:
NS_join(G) is a request by a client to make it a member of group G;
NS_leave(G) is a request to be removed from the membership of G;
NS_resolve(G) is a request to receive the current membership list (in a Resolve_Reply message/event) as it is reflected by the notification service;
NS_sendmsg(data, destination) is a request to reliably send a message (data) to a set of receivers (destination) via the notification service.

Clients of the notification service receive notifications via process events, that indicate the type of the event and the data associated with it. The event associated with the reception of a message via NS (a message that was sent using the *NS_sendmsg* function) is called *NS Recvmsg*.

The notification service contains a failure detector module. Since we assume an asynchronous environment, the notification service is bound to be unreliable in some runs [11], which means it may wrongly suspect correct processes. Since we deal with a service that can be implemented in an asynchronous environment, we do not require the notification service to be accurate. However, we assume that the notification service is always complete [11]. The overall liveness of Xpand does not depend on the notification service providing consistent sets, since it communicates with any connected set of clients. Congress [12] fulfills the

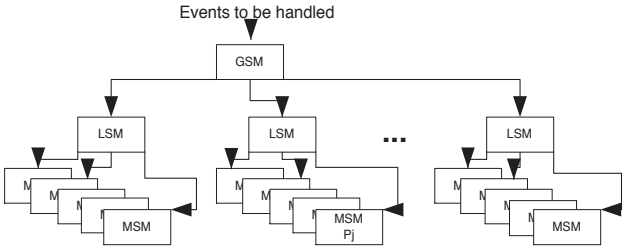


Fig. 2. State Machine Flow.

requirements of the notification service. Other group communication membership services (cf. Spread [4]) can also provide similar notification service.

3.2 Xpand's System Structure

The Xpand group communication system is fully described in [1]. Here we present only the essence of the system and the relevant assumptions. The processes participating in Xpand system are grouped into clusters so that all members of each cluster belonged to the same LAN. The clusters are spread over a WAN. We distinguish between two types of processes: *regular* and *delegate*. The regular processes run the user's application. In each LAN, the delegate is a designated daemon that serves as a representative of its LAN to all the other Xpand clusters.

Although a LAN delegate is replicated for fault-tolerance purposes, only a single delegate is active at each particular moment. This delegate is called an *active delegate*. All the other LAN delegates, called *backup delegates*, serve as warm backups of the active delegate. In the context of this paper, we refer only to a single delegate per LAN and ignore the issue of replacing a failed delegate.

For the sake of simplicity, the membership algorithm is described in this paper in the context of a single invocation of a process, which means that it may join and leave only once. In practice, an instance identifier per process distinguishes among different incarnations of the same process. The relationships between Xpand's components and its environment are shown in Figure 1.

4 Xpand's Ad Hoc Membership Algorithm

Xpand's membership algorithm deals with "views" of a group G. The view at p (meaning the view currently available to a specific process p's) is the current list of connected and alive members of G, as perceived by p. The maintenance of the view of G within p is based on an initial Resolve_Reply event received by p from NS. p updates its view by applying the information received in NS notifications, such as processes' joining or leaving.

Before describing the membership algorithm, we focus on the goals it aims to achieve. The algorithm should allow a new member p_i to join as fast as possible, while maintaining the following properties:

- p_i will start receiving messages from any active member (sender) p_j as soon as possible (given that the sender is alive and transmitting messages).
- For each sender p_j, a new member p_i will receive messages sent by p_j in the same order they were sent by p_i (i.e., FIFO order will be maintained).
- Once p_i receives a message from a specific member, which remains alive and within the same network partition as p_i, p_i will receive all the messages sent by that member from this message on without any gaps. The only situation in which gaps are allowed in the received message stream is when the source is declared disconnected by NS.

The first property will be obtained if a new joining member is notified of the sequence number of the following message from any sending member without any unjustified delay. In order to guarantee the other two properties, there is no need for a global synchronization on sequence numbers that are distributed to any new member with respect to a specific member p_j. This means, for example, that if any two new members p_1 and p_2 are joining simultaneously, a sending member p_j can announce a sequence number $SeqN$ to p_1 and a short time interval later announce $SeqN+k$ to p_2, in case it had managed to transmit k messages during this interval of time.

The asynchrony makes it impossible to describe the protocol in the traditional manner. The protocol can be viewed as an embedded set of state machines executed at the regular member and at the delegate. The top state machine is the *group state machine (GSM)*. That state machine invokes multiple *LAN state machine (LSM)*, one per LAN that contain members in the current view. The LAN state machine invokes *member state machines (MSM)*, one per each member that is listed in the view and is residing within that specific LAN. Figure 2 presents the general flow of control among the different state machines. The GSM receives events and forwards them to the appropriate LAN state machines from which the events are applied to the corresponding MSMs. The events are handled concurrently by the relevant state machines.

For brevity, we included the details of only some of the state machines into the paper. We have chosen to present the group state machine of a regular member and the LAN state machine of a delegate. For the sake of simplicity, we removed handling of various timeouts from our presentation of the state machines.

The membership algorithm uses the set of messages described in Figure 3. All the messages in the figure are messages sent by either delegates or regular members. These messages are sent via the transport services (Section 3).

The ad hoc membership algorithm handles the joining/leaving event, as well as network partitions and network merges. The state machines respond to external events received from NS, as well as to control messages sent by other members of the group (via the corresponding state machines).

The simplified pseudo code of a regular member joining is shown in Figure 4. The code covers the state machine[6] shown in Figure 6 and portions of other two state machines related to this specific case (see GSM-delegate and MSM-

[6] The related states and state transitions in Figure 6 are indicated.

Message Type	From	To	Message role
Force_Join	Regular member	Delegate	Causes Delegate to join a group
Start_Sync	Regular member	Delegate	Causes Delegate to reply with information about group members
Sync_Reply	Delegate	Regular member	Delegate's reply to the Sync_Reply (contains a list of members message sequence numbers as currently known to Delegate)
View_Id	Delegate	Delegate	Causes Receiver to reply with information about its **local** members
Cut_Request	Regular member	Delegate	Causes Delegate to reply with information about specific group members
Cut_Reply	Delegate	Delegate / Regular member	Delegate's reply to the View_Id / Cut_Request messages (contains a list of the requested members message sequence numbers as currently known to Delegate)

Fig. 3. Protocol messages.

delegate in [13])[7]. A regular joining member p_1 is notified about a new member p_2 in the group by receiving either a NS Join message or a Sync_Reply/Cut_Reply message (sent by its own delegate). In either case, p_1 initiates a new member state machine for p_2. In the former case, the MSM goes directly to "active" state, in which messages sent by p_2 will be processed. In the latter case, the NS notification regarding the joining of p_2 has not arrived yet. Thus, the MSM goes into "semi-active". In this state, p_1 waits for the proper NS Join message concerning p_2. When the NS Join message arrives, the MSM goes into "active" state. This complication is caused by the asynchrony resulted from the separation of the membership algorithm within Xpand and the external NS mechanism.

To limit some of the potential race conditions, the Start_Sync message is sent via NS. This ensures that by the time a delegate receives this message via NS, it has already received and processed all the previous NS messages[8], especially those received by the regular member at the moment of sending this request. This doesn't cover the multi-join case that is created when several members join at once, some of them not yet having delegates in the group. In a more complicated case, we may face a situation when members are in the group, but their remote delegates are not, or a situation when the Sync_Reply doesn't include all the necessary information about them. The regular LSM (in [13]) copes with those situations. LSM essentially needs to identify the case and to wait for its resolution, while allowing the sender to begin sending its information. While waiting for the delegate of a LAN to join the group, the member can

[7] E.g., the NS_Join in the code is actually part of the GSM of the delegate.
[8] By the causality assumption of the reliable FIFO comm. layer within the NS.

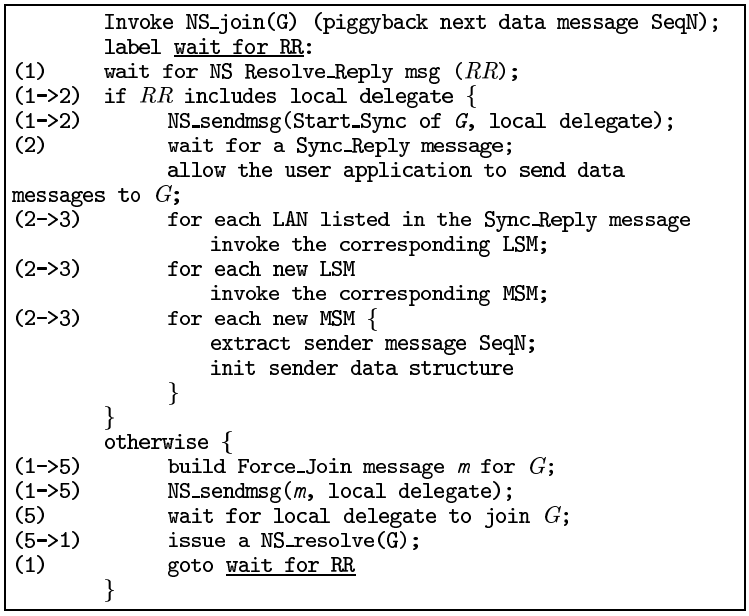


Fig. 4. Highlights of Join Operation at Regular Member.

process further NS messages for LANs on which it has the full information[9]. Other NS messages need to be buffered and processed after the delegate joins and integrates within the group.

As regards the delegate part, a delegate joins a new group when its first local member joins it and "forces" the delegate to join also by sending a Force_Join message. Upon receiving such a message, the delegate invokes a GSM for the relevant group. Upon receiving the resolve reply, the GSM invokes a set of LSMs which, in turn, invokes a set of MSMs per LSM. The delegate GSM and MSM appear in [13]. Below we discuss the delegate LSM (LAN State Machine).

Figure 5 shows the simplified pseudo code of the delegate, executed upon receiving a Force_Join. The code covers the state machine[10] in Figure 7 and *portions* of the other two state machines related to this specific case (see MSM and GSM of delegate in [13])[11]. The Force_Join message causes the delegate to join a specific group. When the delegate joins the group and receives the resolve reply through NS, it needs to spawn LAN state machines per LAN that is represented in that resolve reply message. While the delegate is waiting for the resolve reply message, it may receive View_Id messages or Start_Sync messages

[9] Full information here means acquiring message sequence numbers for each known member in the remote LAN.

[10] The related states and state transitions in Figure 7 are indicated.

[11] E.g., the NS_Join in the code is actually a part of the GSM of the delegate.

```
  label init-group:
        A Force_Join for group G is received from a local member
        NS_join(G);
        wait for NS Resolve_Reply msg (RR);

        split RR into separate LAN lists;
        for each LAN list in RR {
              invoke LAN state machine (Figure 7);
              if remote delegate is NOT listed in the RR
                  wait for remote delegate to join G;

  label peer sync:
(1->2)        NS_sendmsg(View_Id msg, remote delegate);
(2)           wait for Cut_Reply msg corresponding to View_Id msg;
(2->3)        for each member listed in the Cut_Reply msg {
                  invoke the corresponding MSM and within it:
                          extract member message SeqN;
                          init member data structure;
              };
(3)           put LSM into ''active'' state
        } /* for each LAN in the group... */
```

Fig. 5. Highlights of First Join Operation at Delegate.

via NS. Those messages will be buffered and processed later when the resolve reply is received.

When the delegate is already a member of a group and is notified about a new member of its own LAN (via an NS Join notification), it takes the new member message SeqN that is listed in NS Join notification and initializes the new member data structure.

A different case occurs when a remote member joins the group while its own delegate is not a member yet. In this case, the local delegate waits for the remote delegate to join via the NS. Once the remote delegate has joined, the local delegate performs the protocol in Figure 5 starting from peer sync label.

In the above description, we have dealt only with a common case from the overall protocol. A careful use of the NS channel enables us to cope with some of consistency problems. The full protocol is a combination of all the state machines being executed concurrently while responding to the arriving events. Due to the lack of space, we are not considering all the particular issues. The description of the state machines here and in [13] enables to identify some of those issues.

5 Implementation and Performance Results

To test the efficiency of the ad hoc membership algorithm and its implementation, we conducted several tests to measure the effect of membership changes

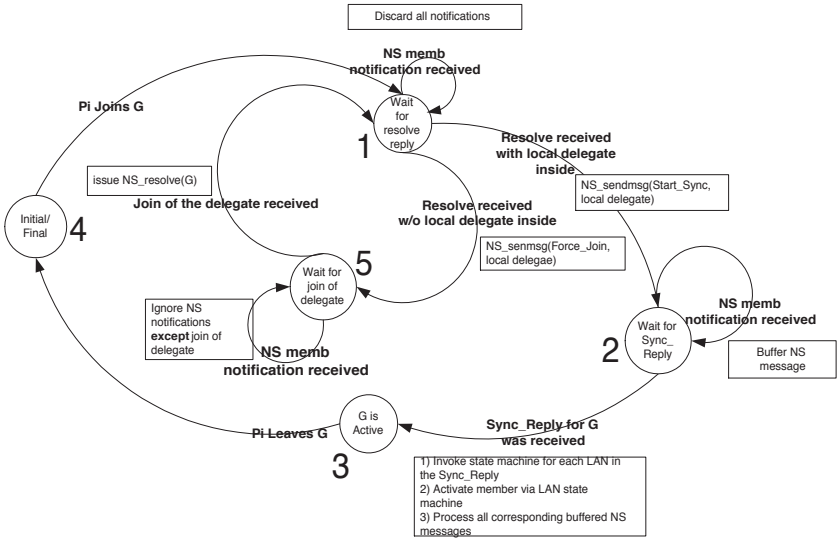


Fig. 6. Group State Machine at Regular Member.

on ongoing message flow. Specifically, we will show its fast_join and smooth_join (Section 2) properties.

The ad hoc algorithm was tested in multi-continent (WAN) setting. The tests included three sites: The Hebrew University of Jerusalem in Israel (HUJI), the Massachusetts Institute of Technology (MIT) and the National Taiwan University (NTU). We had estimated the round trip times and the loss percentage between these sites, in order to have the characteristics of the network interconnecting them. The round trip times measured between MIT and HUJI and between MIT and NTU were both about 230ms. The round trip time between HUJI and NTU was about 390ms. The loss percentage was not as persistent as that of the round trip times, varying from 1 percent to 20 percent[12].

In all the three sites, the machines used were PC computers running the Linux operating system. As there is no IP multicast connectivity among these sites, a propriety application layer multicast (ALM) mechanism was used [1]. The obtained message distribution tree is presented in Figure 8(a). HUJI was selected to be the tree root by the algorithm used in the dynamic ALM tree construction. In all the tests the senders emitted a message every 100ms.

Fast_Join

In the first test, we measured the time it takes a new joiner to start receiving messages from active sources, the participants being 4 senders at HUJI (S1-S4) and one sender at NTU (S5). There were two joiners, one at MIT and one at

[12] In some cases, a higher loss percentage was observed when the interval between the ping (ICMP) messages was less then 100ms.

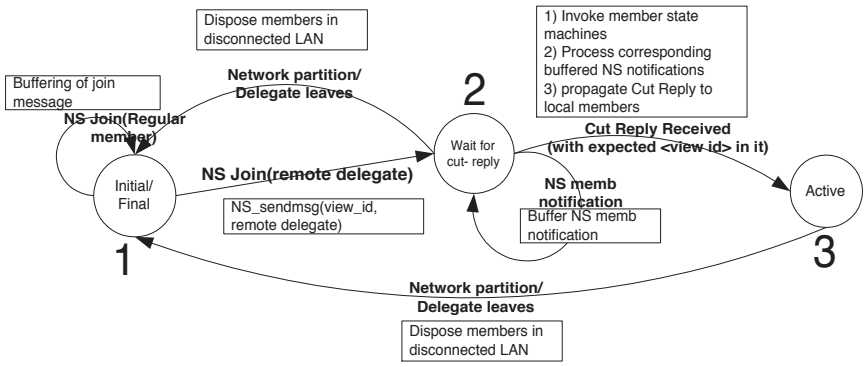
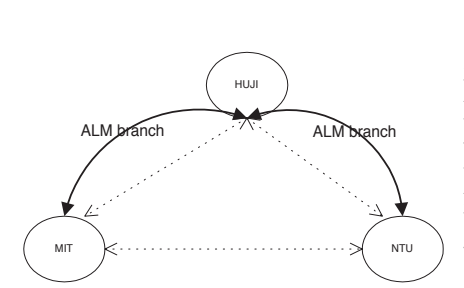
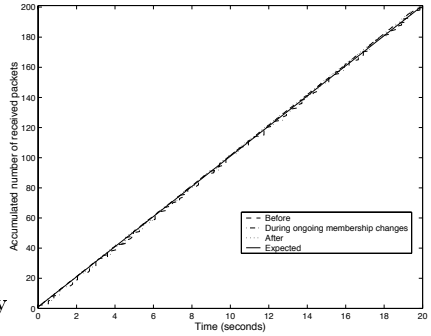


Fig. 7. LAN State Machine at Delegate.



(a) Application Layer Multicast tree topology



(b) Continuous test.

Fig. 8. The Basic Layout.

HUJI (S6). Both joined the group during the test and started sending messages right after joining the group.

The graph in Figure 9(a) shows that once the joiner at MIT (which recorded the events) receives the proper event notification (Sync_Reply message), it begins receiving messages from all current senders within a short time (23ms). It takes the joiner longer time to begin receiving from the another joiner. This extra time is a result of the difference between the arrival of the Resolve_Reply notifications between MIT and HUJI.

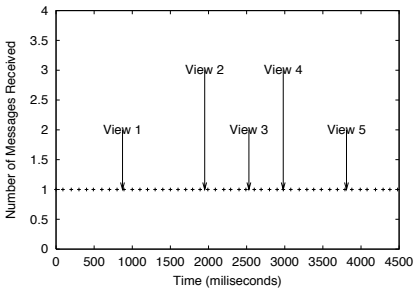
To evaluate the impact of our results, it should be mentioned that in Moshe [7], in a similar scenario, the minimal time should be at least one and a half round trip, which amounts to at least 600ms.

Smooth_Join

In this test, we measured the impact that a set of joiners might have on the ongoing message flow. The sender was at HUJI, the receiver at MIT, and during the test, 4 processes joined at HUJI and one at NTU. All the joiners left later

Event	Time (milliseconds)
NS Resolve	T=0
Sync Reply	T+50
Msg from S1 (HUJI)	T+60
Msg from S2 (HUJI)	T+60
Msg from S3 (HUJI)	T+68
Msg from S4 (HUJI)	T+68
Msg from S5 (NTU)	T+73
Msg from S6 (HUJI)	T+253

(a) First message receiving time.



(b) Simultaneous join impact.

Fig. 9. Impact of membership changes.

on. The tests show that the impact on the message flow was negligible 9(b). Messages are still received every 100ms.

In the virtual synchrony implemented by Keidar et. al [7], the message flow needs to be stopped for at least one round-trip time (in the best possible scenario). In our specific setting it should have taken at least 390ms.

Continuous behavior

The third test measured the behavior of the ad hoc membership implementation over a long period of time, during which the loss rate was rather high. In this test, the sender was located at HUJI, the receiver at MIT, while the process at NTU and the processes at HUJI joined and left repeatedly. The tests show that the behavior of the system during the periods with no membership changes and the periods with such changes is almost identical 8(b), showing the efficiency of the algorithm.

Discussion

All the tests described above proved that the ad hoc approach enables processes to join the group promptly, with minimal impact on ongoing message flow. Applications that do not need strong characteristics will face the minimal impact we observed. Applications that require stronger semantics will still need to wait for full synchronization, as measured by Keidar et. al [7].

6 Related Work

Congress [12] and Maestro [14] were the first to observe that separating maintenance of membership from a group multicast will better enhance scalability of fault-tolerant distributed applications. This separation was later adopted by researchers who addressed the WAN environment ([7, 15, 16]). InterGroup [5] presents another WAN approach to address scalability.

Several research projects in the past sought to relax the semantics of the middleware for distributed applications ([4, 6, 14, 16–21]). Our work takes advantage of both approaches, in order to find a better balance between efficiency, scalability and guaranteed semantics.

Recently, new implementations [22, 23] of reliable multicast have appeared, whose protocols use peer-to-peer overlay systems. Those systems scale for large group, while members are not required to keep group membership information, which might be critical for some applications. An interesting implementation of application layer multicast is Narada project [24]. We are considering using the latter approach as an application layer in Xpand.

7 Conclusions

The focus of this paper is to address the needs of a class of distributed applications that require high bandwidth, reliability and scalability, whilebut not requiring the strong semantics of current distributed middleware solutions. Since current middleware cannot scale well when it is required to guarantee the strong semantics, there is a need to identify a better tradeoff between the semantics and the efficiency.

The ad hoc membership algorithm that we have developed, together with its implementation, present such a tradeoff. The performance results prove that our approach is feasible and can scale well.

The implementation shows that it is possible to integrate an external membership service with a hierarchical system for message distribution. We believe that other systems with hierarchical architecture and/or external membership service may apply similar techniques to their algorithms. A reliable multicast based on forward error correction is a simple one-to-many application. However, the rest of the applications listed in Section 2.1 need better reliability and coordination than our approach offers. Future research is intended to provide better characterization of these classes.

References

1. T. Anker, G. Chockler, I. Shnaiderman, and D. Dolev, "The Design and Performance of Xpand: A Group Communication System for Wide Area Networks," Tech. Rep. 2001-56, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, August 2001, See also the previous version TR2000-31.
2. ACM, *Communications of the ACM 39(4), special issue on Group Communications Systems*, April 1996.
3. Leslie Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.
4. Y. Amir, C. Danilov, and J. Stanton, "A Low Latency, Loss Tolerant Architecture and Protocol for Wide Area Group Communication," in *Proceedings of ICDSN'2000*, 2000.
5. K. Berket, Deborah A. Agarwal, P. M. Melliar-Smith, and Louise E. Moser, "Overview of the intergroup protocols," in *International Conference on Computational Science (1)*, 2001, pp. 316–325.
6. Mark Hayden and Kenneth Birman, "Probabilistic Broadcast," TR 96-1606, dept. of Computer Science, Cornell University, Jan 1996.

7. I. Keidar, J. Sussman, K. Marzullo, and D. Dolev, "A Client-Server Oriented Algorithm for Virtually Synchronous Group Membership in WANs," in *20th International Conference on Distributed Computing Systems (ICDCS)*, April 2000, pp. 356–365, Full version to appear in TOCS.
8. G. Chockler, N. Huleihel, I. Keidar, and D. Dolev, "Multimedia Multicast Transport Service for Groupware," in *TINA Conference on the Convergence of Telecommunications and Distributed Computing Technologies*, September 1996.
9. I. Rhee, S. Cheung, P. Hutto, and V. Sunderam, "Group Communication Support for Distributed Multimedia and CSCW Systems," in *17th Intl. Conference on Distributed Computing Systems*, May 1997.
10. Tal Anker, Danny Dolev, and Idit Keidar, "Fault Tolerant Video-On-Demand Services," in *Proceedings of the 19th International Conference on Distributed Computing Systems, (ICDCS'99)*, June 1999.
11. Tushar Deepak Chandra and Sam Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.
12. T. Anker, D. Breitgand, D. Dolev, and Z. Levy, "CONGRESS: Connection-oriented group-address resolution service," in *Proceedings of SPIE on Broadband Networking Technologies*, November 2-3 1997.
13. T. Anker, I. Shnaiderman, and D. Dolev, "Ad Hoc Membership for Scalable Applications," Tech. Rep. 2002-21, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, April 2002.
14. K. P. Birman, R. Friedman, M. Hayden, and I. Rhee, "Middleware support for distributed multimedia and collaborative computing," in *Proceedings of the Multimedia Computing and Networking (MMCN'98)*, 1998.
15. T. Anker, G. Chockler, D. Dolev, and I. Keidar, "Scalable group membership services for novel applications," in *Networks in Distributed Computing (DIMACS workshop)*, Marios Mavronicolas, Michael Merritt, and Nir Shavit, Eds. 1998, vol. 45 of *DIMACS*, pp. 23–42, American Mathematical Society.
16. Jeremy B. Sussman, Idit Keidar, and Keith Marzullo, "Optimistic virtual synchrony," in *Symposium on Reliability in Distributed Software*, 2000, pp. 42–51.
17. D. Powell, *Delta-4: A Generic Architecture for Dependable Distributed Computing*, Springer-Verlag, 1991.
18. D. Dolev and D. Malki, "The Transis Approach to High Availability Cluster Communication," *Communications of the ACM*, vol. 39, no. 4, April 1996.
19. R. van Renesse, T. M. Hickey, and K. P. Birman, "Design and Performance of Horus: A Lightweight Group Communications System," TR 94-1442, dept. of Computer Science, Cornell University, August 1994.
20. R. Friedman and R. van Renesse, "Strong and weak virtual synchrony in Horus," in *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems, (SRDS'96)*, October 1996.
21. Katherine Guo and Luis Rodrigues, "Dynamic Light-Weight Groups," in *Proceedings of the 17th International Conference on Distributed Computing Systems, (ICDCS'97)*, May 1997.
22. S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant widearea data dissemination," 2001.
23. Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel, "SCRIBE: The design of a large-scale event notification infrastructure," in *Networked Group Communication*, 2001, pp. 30–43.
24. Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communication (JSAC)*, To appear.