J. Parallel Distrib. Comput. 68 (2008) 1081–1096

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

GMAC: An overlay multicast network for mobile agent platforms

Pablo Gotthelf, Alejandro Zunino*, Cristian Mateos, Marcelo Campo

ISISTAN Research Institute. UNICEN University. Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina

ARTICLE INFO

Article history:
Received 9 October 2006
Received in revised form
7 February 2008
Accepted 2 April 2008
Available online 11 April 2008

Keywords:
Multicast networks
Overlay networks
Mobile software agents
P2P
Computer networks

ABSTRACT

The lack of proper support for multicast services in the Internet has hindered the widespread use of applications that rely on group communication services such as mobile software agents. Although they do not require high bandwidth or heavy traffic, these types of applications need to cooperate in a scalable, fair and decentralized way. This paper presents GMAC, an overlay network that implements all multicast related functionality – including membership management and packet forwarding – in the end systems. GMAC introduces a new approach for providing multicast services for mobile agent platforms in a decentralized way, where group members cooperate in a fair way, minimize the protocol overhead, thus achieving great scalability. Simulations comparing GMAC with other approaches, in aspects such as end-to-end group propagation delay, group latency, group bandwidth, protocol overhead, resource utilization and failure recovery, show that GMAC is a scalable and robust solution to provide multicast services in a decentralized way to mobile software agent platforms with requirements similar to MoviLog.

© 2008 Elsevier Inc. All rights reserved.

1. Introduction

Mobile agents are software entities able to autonomously migrate their execution to achieve their users' goals [12]. The running state of a mobile agent can be saved, moved to the location where a required resource is located, and be restored. As a consequence, agent–resource interactions are local instead of remote, thus reducing network usage. Besides efficient network usage, mobile software agents provide advantages such as scalability, reliability and disconnected operations [18].

Special platforms are required to execute mobile agents by supplying them with the resources and mechanisms they need. Although mobile software agents reduce network load, their platforms often need to send small multicast messages to the other platforms to coordinate themselves. For example, platforms such as MoviLog [40,20] require multicast for providing group communication services to agents and for managing their mobility. It is also desirable that these platforms cooperate autonomously and fairly, and do not rely on hierarchical or centralized components, as they usually belong to independent organizations.

The lack of proper support for communicating a large number of distributed mobile agent platforms in a decentralized way has been a hurdle to achieving true scalability of mobile software systems in real settings. As a result, an easily deployable, decentralized and scalable multicast service is a major requirement for the success of mobile software agents in the Internet.

In order to support IP multicast on the Internet, the MBONE (Multicast Backbone) has been built. The MBONE is a virtual network extended across the Internet that supports IP multicast traffic [8]. Despite the need for multicast services, the usefulness of the MBONE is still limited [7]. Furthermore, the MBONE does not reach all Internet users, since neither all routers, nor ISPs (Internet Service Providers) support it.

An alternative for multicast communications that do not require special routers are overlay multicast networks [24,6]. Overlay networks, instead of being supported at the network level, are supported by user-level applications, relying only on unicast as the subjacent service. In this way, neither special routers nor extra ISP involvement is required. At present, many research efforts are being made to provide multicast services through overlay networks [6,33]. Different alternatives were built subject to the particular application requirements involved in each case.

GMAC has been developed to provide multicast services to MoviLog, taking into consideration the following communication requirements:

- *Burst transmission:* Group members transmit small messages for short periods of time (i.e. no data streaming).
- *Stable groups:* Members are supposed to use the service for long periods of time (i.e. hosts do not join or leave the group very frequently).
- *Solidarity among group members*: Everyone in the group cooperates. Each member is interested in other members receiving group messages, even those not sent by it.

* Corresponding author at: ISISTAN Research Institute. UNICEN University. Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina.
E-mail address: azunino@exa.unicen.edu.ar (A. Zunino).

0743-7315/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.04.002

- *Homogeneous groups and network links:* Members have similar behavior and communication characteristics. Therefore transmission probability, connection resources and communication demands are similar.
- *Unknown Topology:* In contrast with other kinds of networks, it is very difficult to determine or take advantage of the Internet topology.
- *Connection restricted clients:* Multicast services to connection restricted hosts should not be denied, though they may receive a degraded service. These unreachable hosts are restricted by firewalls and Network Address Translators (NATs) that do not allow having open ports or using port-forwarding mechanisms.

Since none of the existing approaches completely meet these communication requirements, we have developed GMAC (Group Management Agent Cast), an overlay multicast network for mobile agent platforms on the Internet. In particular, we limited the scope of our research to MoviLog, a platform for mobile agents for the WWW.

The main contribution of this work is to introduce a new approach of providing scalable multicast services in the Internet, allowing independent entities with low group communication requirements, such as mobile agent platforms, to cooperate in a robust, fair and decentralized way.

The rest of the paper is structured as follows. The next section focuses on the MoviLog platform and its multicast communication requirements. The most relevant related approaches are discussed in Section 3. The GMAC model is described in Section 4. Group formation and recovery algorithms are described in Section 5. In Section 6, we present implementation details and the application programming interface of GMAC. Experimental results and comparisons with related approaches are reported in Section 7. Finally, Section 8 presents the conclusions and future work.

2. MoviLog

MoviLog [39] is a platform for mobile software agents based on the concept of Reactive Mobility by Failure (RMF) [40]. RMF aims at simplifying mobile agent development by allowing the programmer to delegate decisions on when and where to migrate a mobile agent based on its resource requirements. The idea is that when a mobile agent needs – at some point of its execution – a resource that is not available at the local site, RMF acts by transferring the agent to a site where the required resource is offered. As a consequence, mobility is transparent to the programmer, agents are smaller and migrate faster [40].

MoviLog relies on servers called MARlets, which provide the run-time support for executing mobile agents. In addition, MARlets provide services for sharing resources such as programs, data or any other agent accessible asset. The backbone of the RMF run-time support is a distributed multi-agent system composed of non-mobile agents called Protocol Name Servers (PNS). This system is in charge of managing information about resources available at each host capable of executing mobile agents. Each PNS can announce to others about new available resources, resources that are no longer available and useful information for managing mobility such as network links status, CPU and memory usage at each host, etc.

In order to manage this information, PNSs must cooperate using a multicast communication mechanism. Up to now, this communication was supported through message broadcasting, limited to local area networks or sequential unicast, hence causing suboptimal network utilization due to excessive network traffic. These issues are not easy to address because most approaches for providing multicast services rely either on special routers, are designed for single sender media streaming, or are unable to handle large groups, hosts behind firewalls or network address translators (NATs) in a decentralized way. In addition, existing approaches demand more network bandwidth for managing the overlay than the messages sent by the PNSs. At this point it is worth noting that PNSs are stationary thus they do not require multicast with support for mobile hosts. As a consequence, a multicast support to meet MoviLog requirements was designed and implemented. The next section reviews the most relevant related work.

3. Related work

Due to the problems that IP Multicast presents [7], specially its low adoption, and the increasing number of applications requiring multicast services on the Internet, several alternatives have been proposed. Some of the most relevant approaches are:

- Overcast [15] provides support for diffusion of information such as video and audio streaming, for a single sender, by disseminating servers across the Internet.
- YOID [11] and HMTP (Host Multicast Tree Protocol) [36] build distribution trees in order to join IP multicast islands. In YOID members must discover and select a parent to join the group. In HMTP, UDP tunnels are used between designated members to connect IP multicast islands.
- Scattercast [5] is also broadcast oriented, but based on special application-level proxies called SCXs spread across the Internet.
- REUNITE [33] uses recursive unicast trees to implement multicast services. REUNITE uses special routing tables in its protocol. One drawback of REUNITE is its reliance on special routers thus it is very difficult and expensive to deploy. However, REUNITE can be incrementally deployed in the sense that it works even if only a subset of the routers implement it.
- ALMI [24] creates a MST (Minimum Spanning Tree) as an overlay structure among the hosts forming a group. In order to build the MST, latency measures between hosts are taken. The main drawback of this approach is that it depends on a centralized component to generate and maintain the MST. In addition, ALMI is restricted to small groups, since the cost of generating the MST increases exponentially as the size of the group grows.
- NARADA [6] improves ALMI by achieving decentralization, though it is still restricted to small groups (less than 200) due to its exponential protocol overhead costs.
- NICE [3] and LARK [17] reduce NARADA protocol overhead in order to achieve scalability, but still focus on data streaming and suffers from long failure recovery delays.

Each one of these alternatives offer multicast support for different communication requirements, based on the specific nature of the multicast communications they intend to support. Overcast and Scattercast have been created with the objective of providing broadcast services, such as audio and video streaming. As a consequence, these approaches are best suited for communications with a single sender and multiple receivers. On the other hand, ALMI and NARADA allow all the group members to send data.

YOID and HMTP achieve a logarithmic scaling behavior. However, they are intricate and inefficient. For example YOID employs expensive and complex techniques for loop detection and avoidance. In HMTP nodes are constantly looking for a better parent, producing a considerable overhead.

REUNITE, Overcast and Scattercast are not well suited for multiple senders, which is the case of MoviLog. In Addition, they depend on specific routers spread across the Internet, therefore, they share some of the problematic characteristics of the MBONE.

Though ALMI and NARADA implement diffusion groups by creating overlay networks, both still suffer scalability problems restricting them to small groups. NICE and LARK achieve better

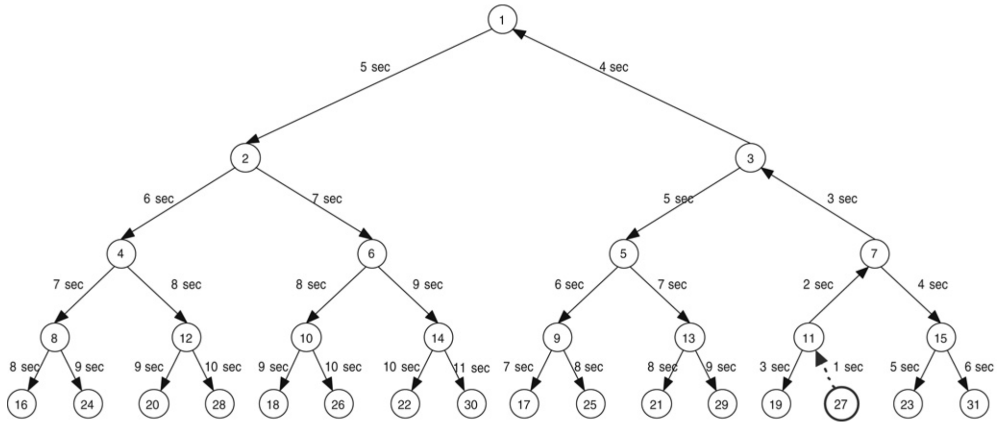


Fig. 1. Example of message forwarding in GMAC.

scalability, but still suffer from high failure recovery delays and protocol overhead problems.

Nowadays the notion of P2P (Peer to Peer) networks [31] is attracting the attention of the Internet community. Some of these approaches use the TTL [1] flooding method for spreading messages [28]. As this scheme lacks efficiency, heuristics to avoid flooding, such as random walk [13] have been adopted. However, as mobile agent platforms have specific communication requirements, for instance, in MoviLog reaching all group members is mandatory, these unstructured approaches are not adequate. Distributed Hash Tables (DHT) [2] are used to structure P2P overlays forming a virtual space where nodes and content are unambiguously identified, thus messages can be forwarded through the corresponding path. Over this P2P scheme, some alternatives were built to provide multicast services. For example CAN [25] still uses flooding, but in a directed way. Scribe [4] and Bayeux [38] were built on top of Pastry [30] and Tapestry [37] respectively, building a distribution tree per group. Although these approaches are scalable, they still do not completely apply to MoviLog requirements. In CAN, despite loops are avoided, duplicate messages are not fully avoided as the same message may arrive to a host more than once. Bayeux and Scribe form distribution trees assigning an identifiable node as the root of a group, and paths are built towards it. Their main drawback is that they do not make a fair distribution of duties, nodes may become transmission bottlenecks, and even nodes that are not members of a group may participate forwarding its messages.

Mobile agent platforms, and MoviLog in particular, rely on group communication both for agent communication and mobility management. In MoviLog each host of the network is able to announce available resources such as code, data or services. MoviLog multicast messages are small and sporadic, thus in order to reduce the network traffic, it is essential that the overlay management do not demand more network bandwidth than the data messages themselves. In addition, MoviLog requires services with support for multiple senders in a decentralized and easy deployable way. Furthermore, as any mobile software agent platform, it is desirable to support groups as large as possible.

To sum up, since none of the previous approaches provide enough scalability, robustness, deployability and failure recovery, in a fair and decentralized way, with support for multiple senders, we designed GMAC (Group Management Agent Cast) to cope with the requirements imposed by MoviLog.

[1] Time To Live: A message only traverses a limited number of hops before it is removed from the network.

4. GMAC model

GMAC [14] is an application level multicast infrastructure, based on a binary tree as overlay structure representing a group, where each node of the tree corresponds to a host in a group and the links between them are unicast connections.

In order to provide multicast functionality, each host in a group sends data only to their neighbors in the binary tree. These, in turn, retransmit the received data in the same manner, relieving the transmitting host. In this way, every member retransmits data to other two group members at most, and hence consuming similar network resources. As we will explain in Section 4.2, this approach has been chosen since GMAC provides multicast services only for group coordination purposes, while applications running on top of it, such as mobile agent platforms, will be the ones using the network in an intensive manner, for example for migrating software agents.

This data flow scheme works by spreading messages over the tree. Leaf nodes do not retransmit messages, inner nodes retransmit messages to other two group members (at most) and the root just forwards messages once (receives from one child and forwards to the other child). Some properties of GMAC can be appreciated by comparing it with sequential unicast, where each group member has to send a copy of the message to all the other group members. Fig. 1 depicts an example where in a group of 31 members, the node labeled 27 sends a group message. Assuming that every message transmission between unicast links takes one second (the unicast links have the same bandwidth and latency), GMAC would take 11 s for every node to receive the message, while sending a message to each member would require 30 s.

It is worth noting that when doubling the number of group members in the above example, GMAC would take 14 s (only 3 s more), as it has a logarithmic behavior, whereas sequential unicast would demand 61 s to reach all group members.

When assuming that each unicast link takes one second to transmit a message, the total time required to send a group message with the sequential unicast approach would be $n - 1$ s, where n is the total number of group members. On the other hand, the time required by GMAC behaves logarithmically with regard to the number of group members. As GMAC uses a balanced binary tree, the worst case arises when a leaf sends a group message. In this case, the message delivery would encompass the longest path of the tree to reach the farthest leaves. Consequently, the overall time (in seconds) required for the worst case is:

$$tw = 3\left(\lceil \log_2 n \rceil - 1\right) - 1. \quad (1)$$

The best case would happen when the root sends a group message, thus, the time (in seconds) required for the message to reach all the

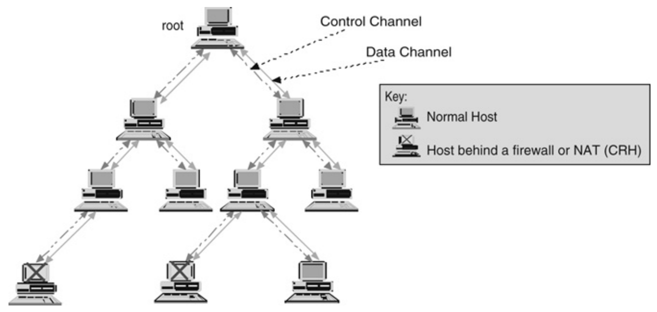


Fig. 2. GMAC tree.

group members would be:

$$tb = 2\left(\lceil \log_2 n \rceil\right), \tag{2}$$

this is, twice the path to the farthest leaf, as the message must be forwarded two times at each host.

4.1. GMAC components

The GMAC Model is based on a binary tree where group members are nodes in the tree. A group member is identified by its IP address and a port number, which are used by the rest of the participants to establish their connections. As Fig. 2 shows, nodes are connected by two unicast links:

- *Control link*: through which control messages, related to the overlay tree building and maintenance, are transmitted.
- *Data link*: through which data messages are sent and forwarded. When a data message arrives, it is immediately retransmitted to other neighbors and sent to the application layer for processing.

Using a double-link schema greatly simplifies the implementation of the model and improves efficiency, although using a single link is also possible.

Algorithm 1 Sending a data message

```
1: dataMessage=application.getMessage();
2: if connectionParent!=null then
3:     connectionParent.sendData(dataMessage);
4: end if
5: if leftChild!=null then
6:     leftChild.sendData(dataMessage);
7: end if
8: if rigthChild!=null then
9:     rightleftChild.sendData(dataMessage);
10: end if
```

Algorithm 2 Receiving and forwarding a data message

```
1: dataMessage=TriggeredDataConnection.receive();
2: if connectionParent!=TriggeredDataConnection and
   connectionParent!=null then
3:     connectionParent.sendData(dataMessage);
4: end if
5: if leftChild!=TriggeredDataConnection and leftChild!=null then
6:     leftChild.sendData(dataMessage);
7: end if
8: if rightChild!=TriggeredDataConnection and rigthChild!=null then
9:     rightleftChild.sendData(dataMessage);
10: end if
11: application.arrivedDataMessage(dataMessage)
```

The entire functionality of GMAC is implemented in each node in a collaborative and decentralized manner, as the responsibilities for message delivery, tree building and recovery are distributed among the members of a group. Ascending control messages are used to provide this functionality in a decentralized way, as only the information below the tree is required in each node. Thus, each host receives control messages from its children, updates its control state and propagates it upward to its own parent. Only changes in the overlay structure, such as host arrival and departure, will trigger these control messages. Algorithms 1 and 2 describe the data message propagation scheme. All algorithms use Java notation with variables denoted in italics. When a message arrives, the node forwards it before delivering it to the application layer.

GMAC also supports hosts with connectivity restrictions. This kind of hosts are behind firewalls or Network Address Translators (NATs) that do not allow having open ports or using port-forwarding mechanisms. These hosts will be referred as CRHs (Connectivity Restricted Hosts) as they are not able to accept incoming connections and hence are incapable of communicating between each other. GMAC only supports CRHs as leaves of the binary tree, limiting them to at most $\lfloor n/2 \rfloor + 1$, where n is the total number of group members. Studies with eDonkey and Gnutella reveal that as many as 36% of the hosts may be CRHs, claiming it as a real problem that needs to be addressed [34]. As GMAC allows 50% of the nodes to be CRHs this problem is correctly addressed.

There is also a component common to all groups called GMAC registry. The main property of the GMAC registry is that it is globally accessible in the Internet, thus works as a "meeting place" for hosts wishing to join a group, as it provides them with the information needed to join any specific group. This information consists of the IP address and port belonging to the current root of a group. The GMAC registry is explained in more detail in Section 5.3.

4.2. Overlay structure

To support multicast communications, most approaches use optimized overlay structures, such as a Minimum Spanning Tree (MST), based on each host connection properties. On the other hand, GMAC uses a binary tree for the following reasons:

- *Scalability:* Generally the cost of keeping an optimized structure, like a Minimum Spanning Tree, is greatly increased as the number of group members grows. As a consequence, MSTs are usually confined to groups with a limited number of members. Examples are ALMI and NARADA.
- *Dynamism of the environment:* Using an optimized structure presupposes that members have dissimilar connection capabilities, and the overlay structure is build based on those parameters. However if those parameters change very often, the structure should be re-optimized frequently. Therefore an optimized structure usually is not suitable in very dynamic environments.
- *Fairness:* Generally the Internet connection in a host is shared by several applications. For the case for GMAC, as other applications will be using the network resources in a more intensive manner, it is not possible to consider that a host with a better connection can be used as a group retransmitter, or burdened with extra tasks. Accordingly, GMAC guarantees that each group member will retransmit data to other two members at most.
- *Decentralization:* The computation used to build and maintain an optimized structure generally relies on a central component, and is not easy to decentralize. In contrast, GMAC uses a self-organizing approach, since all members have the same responsibility and cooperate in the formation and maintenance of the tree structure in a decentralized way.
- *Fault tolerance:* As GMAC does not depend on a central component, a failure in one host will not jeopardize the group. The failure recovery cost is often high in optimized structures, and may imply structure reorganization, which is a time and resource consuming process.

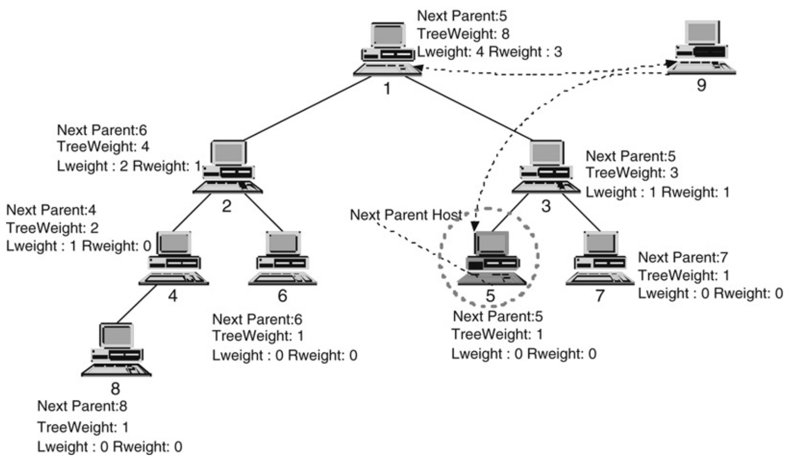


Fig. 3. Improved Join Heuristic.

- *Protocol overhead:* Optimized overlay structures often achieve better resource utilization than unoptimized overlays. However, the generation and maintenance of an optimized structure require performing several measurements, which also use network resources. Considering MoviLog requirements, GMAC avoids taking these measurements because they might use more network resources than MoviLog.

For these reasons, GMAC uses a binary tree without considering each node capabilities in particular. Although having a higher tree degree may improve latency, the overall group throughput would be reduced. In other words, to forward a message in a k-ary tree, internal nodes would have to divide their available bandwidth by k receivers. On the other hand, external nodes (leaves) do not act as message forwarders, as they only transmit their own messages. Since the ratio between the number of internal and external nodes in a k-ary tree with n internal nodes is:

$$\frac{\text{Internal nodes}}{\text{External nodes}} = \frac{n}{n(k-1)+1} \simeq \frac{1}{k}, \qquad (3)$$

the more the tree degree k, the more the load imbalance between internal and external nodes, this is, having less internal nodes which also are more loaded forwarding messages. It is important to notice that in a binary tree ($k = 2$) the number of internal nodes is maximized, being approximately equal to the number of external nodes: $n/(n+1)$. Accordingly, to achieve a logarithmic distribution behavior, GMAC relies on a binary tree as overlay structure.

The next section describes the mechanisms for group formation and maintenance of GMAC.

5. Group management

The general idea to achieve decentralization is that when a node receives a join request it incorporates the requesting node as a child. If there is no room at that node (i.e. it already has two children), it will delegate the join request to its least weighted child (i.e. the one with the smallest subtree). Therefore, a host wishing to join a group will descend the tree until it is inserted as a leaf. As the overlay structure is a binary tree, a host joining a group will have to traverse at most $\log_2 n$ nodes, being n the total number of nodes in the tree.

In the following subsections a more complex join heuristic, used by GMAC to reduce reconnections and allow connection restricted hosts (CRHs), is explained.

5.1. Improved join heuristic

The main idea is that each host in the tree knows where the next connection point down the tree is, so as to delegate an incoming host directly to its joining point node.

In order to provide such functionality, a variable called *nextParent*, consisting in the IP address and port of the node corresponding to the next connection point, is held at each host. The *nextParent* value in a node is determined by the control messages received from its children.

In order to join a group, a node follows Algorithm 3. In addition, each node in the tree follows Algorithm 4 to deal with join requests.

Algorithm 3 Joining a Group

```
1: rootIPandPort=GroupRegistry.getRootIPandPort
   (groupName,password);
2: if rootIPandPort==myIPandPort then
3:   connected=true //I am the root, the GMACRegistry has
     published my IP and port as the root.
4: else
5:   connectionOutgoing=connectToHost(rootIPandPort, groupName,
     password);
6:   if connectionOutgoing.isAcceptedAsChild()==true then
7:     connected=true;
8:     connectionParent=connectionOutgoing;
9:   else
10:    while connected==false do
11:      nextParentIPandPort=connectionOutgoing.getNextParent-
         IpAndPort(); //the nextParent of the least weighted
         child
12:      connectionOutgoing.EndConnection();
13:      connectionOutgoing=connectToHost(nextParentIPandPort,
         groupName, password);
14:      if connectionOutgoing.isAcceptedAsChild()==true then
15:        connected=true;
16:        connectionParent=connectionOutgoing;
17:      end if
18:    end while
19:  end if
20: end if
```

An example of the improved heuristic is shown in Fig. 3. The *nextParent* values are determined in each node as follows:

- Node 8 is a leaf of the tree, thus its own IP address and port is used as the *nextParent* value. Node 4 does the same thing,

Algorithm 4 Attending a Joining request

```
ConnectionIncoming=ReceiveConnectionRequest();
if leftChild==null then
    leftChild=ConnectionIncoming
    leftChild.sendConfirmationToChild();
else
    if rigthChild==null then
        rightChild=ConnectionIncoming
        rightChild.sendConfirmationToChild();
    else
        ConnectionIncoming.sendConnectoToHost
        (this.nexparentIPandPort)
    end if
end if
```

as it has only one child (i.e. it has room for one more host). Node 2 updates its *nextParent* with the information received from its children, selecting the *nextParent* information from node 6 (where the least weighted subtree is) and sends this information to its parent (node 1).

- Node 3 sets its *nextParent* with the value received from node 5 (in equal conditions selects by default the value from the left) and sends it to node 1.
- Node 1 will also set its *nextParent* to the one received from its least weighted subtree, in this case, the IP address and port of node 5, (the *nextParent* received from its right child). When a new host (node 9) wanting to join the group arrives, it will be immediately forwarded to the node 5, which will incorporate it as a leaf.

Each node stores the *nextParent* and tree weight values received from its children. With these values, a node calculates its own *nextParent* and tree weight values and sends them to its parent. The goal of this process is to summarize the subtree state bellow each node. When the overlay structure changes, the affected nodes will have to recalculate their own tree weight and *nextParent* values and, in case the summarizing values have changed, a chain of ascending control messages will be triggered.

5.2. Enabling Connection Restricted Hosts (CRHs)

To allow hosts behind firewalls or NATs, a slight extension to the previous heuristic is required to allow them only as leaves of the overlay tree.

When all the leaves of a subtree are CRHs hosts, it will not be possible to incorporate more hosts of this kind. As a consequence, a new boolean variable called *allowCRHs* is stored by each node to know whether a subtree is capable of accepting join requests from CRHs. This information, together with the *nextParent* and tree weight values, is packed in control messages. To keep CRHs as leaves, when a host having both children and at least a CRH gets a connection request from a non-CRH, it will incorporate it by performing a rotation. As depicted in Fig. 4, a rotation is performed as follows:

(1) It will ask its CRH leaf to connect to the new incoming host.
(2) The CRH leaf will reconnect to the new incoming host.
(3) The new host will be accepted as a child in the place where the CRH was.

As a consequence, every host having a CRH child will set its *nextParent* variable to his own address, as a non-CRH will always be incorporated via a rotation. Algorithm 5 describes the full heuristic for allowing all types of hosts, including the rotation of a CRH child by replacing it with a standard joining host.

It is worth mentioning that in GMAC groups are formed incrementally. Consequently, CRHs join-requests will be refused if the current tree does not support more CRHs. Nevertheless, the application using GMAC could implement a mechanism where CRHs retry to connect after a certain amount of time, waiting for a non-CRH connection or a CRHs departure. An example of the full heuristic, allowing CRHs, is shown in Fig. 5:

- Hosts 4, 6, 7 and 8 are CRHs.
- Both children of Node 2 are CRHs, thus it sets *allowCRHs* to false (it is not capable of accepting more CRHs).
- Node 5 sets *allowCRHs* to true, as it only has one child.
- Node 3 gets this information and sets *allowCRHs* to true, as node 5 is able to accept join requests from CRHs. Nevertheless,

Algorithm 5 Handling a Join request, full heuristic.

```
ConnectionIncoming=ReceiveConnectionRequest();
if leftChild==null then
    leftChild=ConnectionIncoming
    leftChild.sendConfirmationToChild();
else
    if rigthChild==null then
        rightChild=ConnectionIncoming;
        rightChild.sendConfirmationToChild();
    else
        if !ConnectionIncoming.isCRH then
            if leftChild.isCRH() then
                leftChild.sendConnectToHost(ConnectionIncoming.
                IPandPort());
                leftChild=ConnectionIncoming;
                leftChild.sendConfirmationToChild();
            else
                if rightChild.isCRH() then
                    rightChild.sendConnectToHost(ConnectionIncoming.-
                    IPandPort());
                    rightChild=ConnectionIncoming;
                    rightChild.sendConfirmationToChild();
                else
                    ConnectionIncoming.sendConnectoToHost(this.-
                    nexparentIPandPort);
                end if
            end if
        else
            if this.allowCRH then
                ConnectionIncoming.sendConnectToHost(this.
                nextparentCRHIPandPort);
            else
                ConnectionIncoming.sendNoMoreCRHsAllowed-
                NotificationToChild();
            end if
        end if
    end if
end if
```

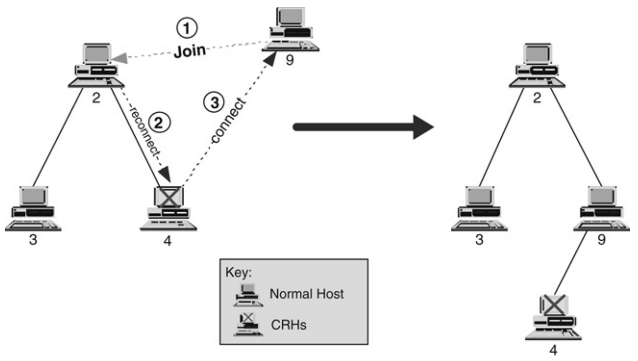


Fig. 4. Rotation of a connection restricted child.

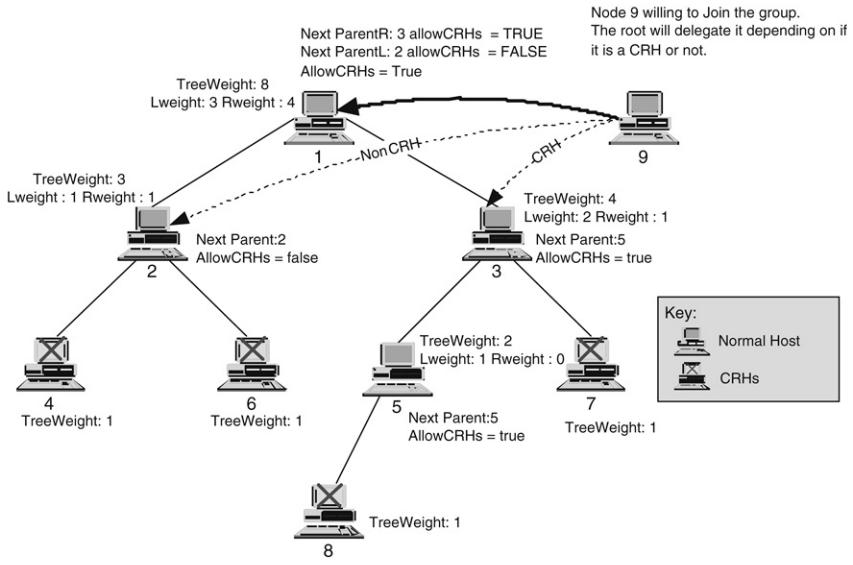


Fig. 5. Heuristic for allowing CRHs.

it sets the *nextParent* value to its own address and port, as a rotation would be needed if a normal host wants to join in the future.

- Node 1 (the root node) will get:
  . From node 2: the address and port of node 2 as *nextParent*, 3 as the tree weight, and that it does not allow any more CRHs.
  . From node 3: the address and port of node 3 as *nextParent*, 4 as the tree weight, and that it allows CRHs.

Thus, when a new host (host number 9) wants to join the group, it will send to node 1 (the root) a connection request, which will be delegated depending on whether the new host is a CRH or not.

It is important to notice that when the *allowCRHs* values of a node's children differ, for example node 3 in Fig. 5, it must set its *nextParent* value to its own address and port, as it will have to decide where to delegate join requests depending on the connectivity capabilities of the incoming host.

Even though others solutions are possible, allowing CRHs as leaves of the tree leads to a simple, clean and convenient solution. Others possibilities, such as having levels of intermediate CRHs (where CRHs and non-CRHs alternate) would add extra complexity to the joining and failure recovery algorithms. In addition, this would admit at most 66.7% CRHs, compared to 50% of having CRHs only as leaves. Other possible approach is NAT traversal [29, 10,9], which enables CRHs to connect directly to each other. However, this approach was avoided as it could bypass local network restrictions and policies.

5.3. GMAC registry

As explained earlier, the main requirement of the GMAC registry is that it has to be globally accessible in the Internet, working as a bootstrap or starting place for hosts wishing to join a certain group. The GMAC registry (Fig. 6) has two responsibilities:

- Publish the roots of the groups: sends the root address and port of a group to a host intending to join it, provided that the correct group name and password is given.
- Replace the published root: provide a mechanism for replacing a published group root in case it fails or leaves.

This component has been materialized as a separate Web application and implemented as a cluster of redundant servers, thus reliability is not compromised.

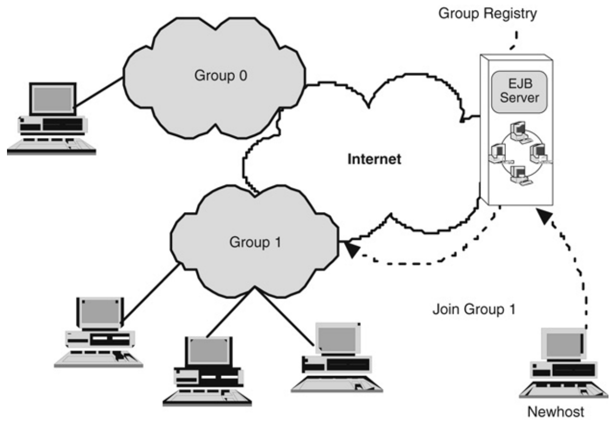


Fig. 6. GMAC registry.

5.4. Failure recovery

GMAC uses a fast failure recovery mechanism, rather than implementing a failure avoidance strategy. When a node fails or leaves a group, the tree must be restructured in order to continue providing multicast support to the rest of the group. This reorganization is done by the remaining members in a decentralized way as follows:

- The parent node, which had the failing node as a child, just closes the connections to it, updates its state information and sends it to its own parent.
- Children nodes, which had the failing node as its parent, must reconnect to the tree by sending a connection request to the root.

For the non-adjacent nodes below the one that failed, reorganization is transparent, as they will be reconnected along with the orphan nodes. In this way, a node failure is handled by reconnecting its children, which is accomplished in a decentralized manner. In addition, tree reorganization involves at most two node reconnections, thus the computational cost of a failure is still logarithmic over the number of nodes. In Algorithms 6 and 7 the failure recovery algorithm is described from the child and the parent point of view respectively. Fig. 7 depicts the GMAC failure recovery mechanism. Suppose node 3 leaves the group. The root, which had node 3 as a child, just releases the connection. Both

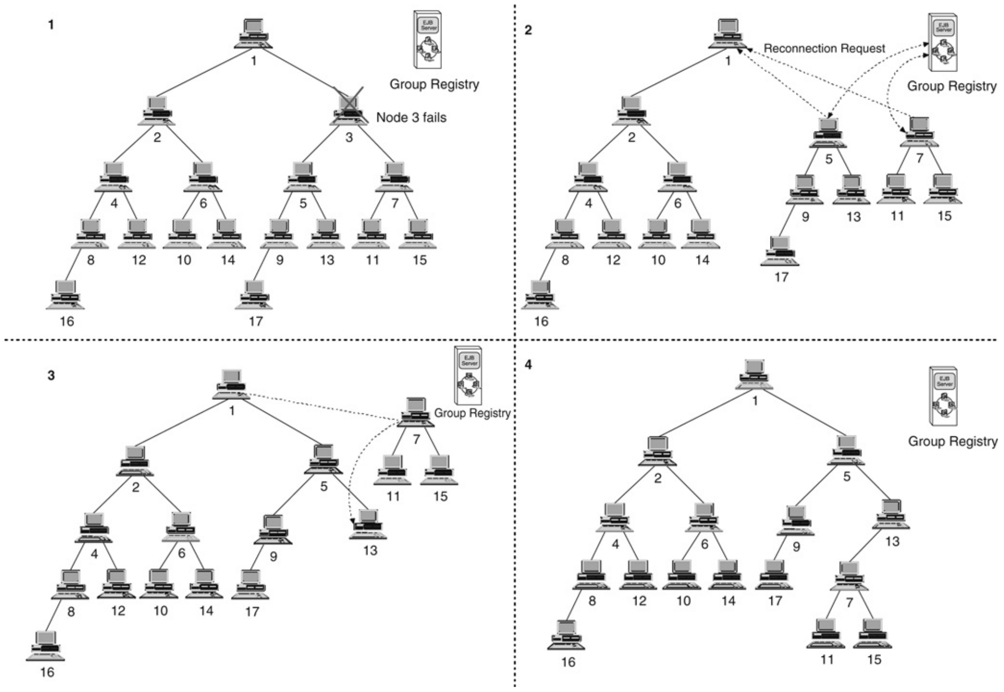


Fig. 7. Failure recovery.

Algorithm 6 Recovering from a parent failure

```
connected=false
while rootFound==false do
  connectionOutgoing=connectToHost(rootIPandPort,groupName,
  password);
  if root not responding after n retries then
    rootIPandPort=GroupRegistry.claimToBeTheRoot(rootIPand
    Port, groupName, password, myIPandPort);
    if rootIPandPort==myIPandPort then
      rootFound=true //claim succeded,(else other one claimed
      first, the new rootIPandPort will be tried)
    end if
  else
    rootFound=true; //the root responded
  end if
end while
goto Algorithm 3 line 6.
```

Algorithm 7 Recovering from a child failure

```
1: controlMessage=connectionChild.receiveControlMessage();
2: updateChildStateInformation(controlMessage);
3: recalculateNextParent();
4: generate and send new controlMessage to parent
```

"orphan" nodes 5 and 7 will reconnect to the root keeping their current children, thus actually a subtree is being reconnected. In this case, the root first handles node's 5 request incorporating it as its right child. Next, the root accepts node 7 request by delegating the join request to node 13.

A special case arises when the node that leaves the group is the root itself, since orphan nodes will fail when attempting to reconnect to it. Therefore, as these nodes notice the root is missing or not responding they will ask the group registry to become the root themselves via a *claimRoot* message. This *claimRoot* message contains the address and port of the supposed failing root in order to let the GMAC registry check if the received root address and port corresponds to the published one (i.e. that it has not been previously changed). Then the GMAC registry will verify that it is really failing, and provided this is true, it will accept the root claim message and publish the new root address and port for the group.

Although the final tree is not completely balanced, the overall overlay performance is not compromised. Besides, the tree will tend to recover its balance as new nodes join the tree. The failure recovery mechanism in GMAC is fast and simple, and neither extra control links, nor extra control state information needs to be maintained in the nodes for failure resilience purposes. As a consequence, GMAC approach is to detect and reconnect disconnected nodes immediately, rather than avoiding nodes disconnection by adding redundancy or complex heuristics.

A *claimRoot* message is rejected if the included root information does not match the GMAC registry published root information. In such cases, a message containing the updated root information is sent back to the claiming node. This will happen after a root fails and both orphan children send a *claimRoot* message to the group registry; the first one to arrive will succeed, while the second will be rejected. However the second one, when rejected, will get the updated root information, which in this case, is the succeeding orphan address and port, i.e. the new root for the group. For this special case, there will be an extra recovery delay, due to the waiting time the non succeeding orphan node must wait until it gets the new root address.

This root substitution method could be seen as the only mechanism in GMAC not fully decentralized, as, in order to join a group, a new node could ask for the root address to an already connected member instead of asking the group registry. As a consequence, only when both, the group registry and the root of a group fail, the root orphans would be unable to replace the root and the group would be divided in two. Although this case in unlikely to occur, group availability must not be compromised. This is why the group registry is implemented as a cluster of redundant servers.

Every time subtrees get disconnected, current messages traversing the tree may be lost. To solve this problem, a first attempt might be to store undelivered messages in buffers every time an adjacent node fails. Next, the reconnected subtree sends a multicast message to contact the nodes that were adjacent of its previous parent. Subsequently, missing messages could

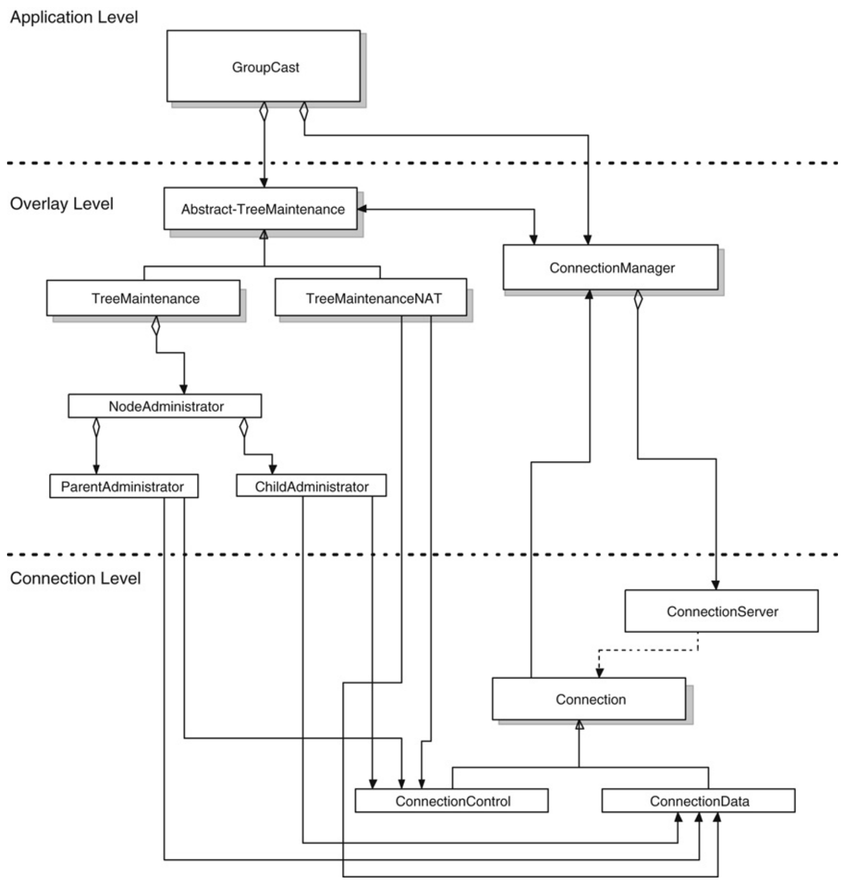


Fig. 8. GMAC Class Diagram.

be exchanged. This approach fails when a parent and a child node fail at nearly the same time. To avoid this problem the reconnected node should ask for all undelivered messages in the tree, generating many replicated messages and possibly congestion. This is why message recovery is optional in GMAC, allowing the specific applications using GMAC to decide which approach to use depending on their communication requirements. For the case of MoviLog, as multicast messages represent changes in the group state, synchronizing the reconnected subtree is enough.

6. Implementation

GMAC has been implemented in Java. Java has many desirable properties that ease application development and deployment. Some of them are multi-threading, inheritance, object orientation, portability and a well known API, just to mention a few. Fig. 8 depicts the simplified UML class diagram of GMAC implementation. The main responsibilities are divided in three classes: `ConnectionManager`, `TreeMaintenace` and `GroupCast`. The whole design was done to favour flexibility. The `GroupCast` class provides a number of methods for using GMAC services, the `ConnectionManager` issues all TCP connections and the `TreeMaintenace` is responsible for keeping the overlay structure. Inheritance was used to distinguish standard hosts from connection restricted ones, and to keep the possibility of adding different implementation strategies open.

The `ConnectionManager` class manages all connections. It has an instance of the `ConnectionServer`, which listens for incoming connections on an open port. Each time a new connection arrives, a new thread handles it and the handshaking process is started. There are 4 types of connections:

- Incoming Connection: A connection a host receives.
- Outgoing Connection: A connection a host starts.
- Control Connection: The connection where ascending control messages are sent.
- Data Connection: The connection where data messages are transmitted.

The handshaking process is done as follows. The `Connection Server` listens on the open port for incoming connections. If the joining request can be satisfied, the incoming connection turns into a control connection, as depicted in the double link schema in Section 4. Once this control connection is established a data connection is created with a similar process.

The `TreeMaintenace` class uses a `NodeAdministrator` class that, in turn, has an instance of the `ParentAdministrator` class and two instances of the `ChildAdministrator` class, one for each child. These instances have the control and data connections for each adjacent node. Besides, they store the *nextParent*, *weigh* and *allowCRHs* values for each subtree (in the case of the child instances).

GMAC has been developed as a Java library to allow developers to use GMAC multicast services easily. For example, to join a group a programmer has to provide GMAC with the group name, password, and the registry IP address and port. The following code shows the steps for joining a GMAC group called GROUPNAME1:

```
GroupCast groupCast = new GroupCast("GROUPNAME1",
"PASSWORD1", gmac.com.ar, 2000);
groupCast.startListeningInPort(3333);
groupCast.joinGroup();
```

Hosts behind NATs or firewalls not allowed by network administration policies to either use port-forwarding or having an open port will be treated as CRHs. For non-CRH, only one reachable open port in the range 1024–65535 must be provided so as not to be considered a CHR by GMAC. This port will be used by the `ConnectionServer` to attend incoming connections.

Once connected, messages can be sent to the group. The following example shows how to send String messages to the whole group:

```
public void sendString(String message) {
    DatagramPacket dp= new DatagramPacket
    (message.getBytes(),message.length());
    this.groupcast.send(dp);
}.
```

The code below shows how to receive messages from the group:

```
public void run () {
  DatagramPacket newPacket;
  while (true) {
    newPacket = this.groupcast.receive();
    this.dealWithPacket(newPacket);
  }
}.
```

Java provides an API for broadcasting and multicasting messages represented as an instance of the `DatagramPacket` class. Many existing applications already use this API. As the previous examples show, in GMAC the `DatagramPacket` abstraction is used for message transfers as well. However, this is only done for easily porting to GMAC applications based on the Java API for broadcasting and multicasting, since TCP connections are used underneath.

The group registry is implemented as an independent application. A cluster of redundant servers based on Enterprise Java Beans [23] is used to address availability. When a node ask the registry to join a certain group given the group name and password, if the given group name does not exist, the group registry sets up a new group with the joining node as the root. The root substitution mechanism is synchronized at group level in other to attend only one claim root at the time, while other requests are processed concurrently. Periodically, the group registry issues a connection to each published group root to check if they are is still running, removing the non responding ones.

7. Experimental results

The alternatives described in Section 3 provide multicast support to applications with very particular multicast requirements. Solutions involving a single sender broadcasting media try to maximize bandwidth, while conferencing oriented applications require low latency as well, though they may allow graceful degradation.

GMAC is intended to allow large groups to communicate by sending small multicast messages. The main features of GMAC are decentralization, scalability and low overhead, while transmission requirements are not considered critical. Nevertheless, it is desirable to analyze and evaluate GMAC performance to determine how it meets MoviLog requirements and to have a better understanding of GMAC behavior.

While unicast connections are usually measured only in terms of bandwidth and latency, for GMAC group communications, the most relevant aspects to consider are:

- *End-to-end group propagation delay:* is the time required for a message to reach the farthest group member.
- *Group latency:* is the maximum delay, from the application point of view, between a sender and the receivers. Within a distribution tree, latency is the longest end-to-end delay between any pair of nodes.
- *Throughput:* is the maximum throughput, from the receiving applications point of view, that a group can provide. Within a distribution tree, nodes with less bandwidth may act as bottlenecks, thus limiting the overall group bandwidth.
- *Protocol overhead:* this metric takes into account the network traffic that does not involve application data, including control messages required to build and maintain the overlay structure.
- *Link stress:* is the number of duplicate packets on the same physical link.
- *Network distance:* At the application level, network distance is generally related to latency, assuming that more latency implies more cost.
- *Failure recovery:* the time required by the overlay to recover from failures in nodes.

Several simulations were made in order to evaluate GMAC. GMAC was compared with sequential unicast and two MSTs (Minimum Spanning Trees) variants, one maximizing the bandwidth and the other minimizing latency. This could be seen as the approaches adopted by NARADA and ALMI, respectively. It is important to note that such MST trees are used as theoretical boundaries for the metrics they optimize. No other overlay tree can achieve better results for a specific metric than a MST built under that measure. Moreover, these MSTs were built within an static environment. Note that it would be very difficult to build them in real conditions, since it would be necessary to know the network topology and its metrics in advance, and to assume they are static, something not likely to happen in the Internet.

As the implementations of the alternatives described in Section 3 are not always available, or they do not support groups with similar size ranges as GMAC, the number of possible experimental comparisons was limited. Nevertheless, some comparisons with approaches such as NARADA, NICE and LARK were made when data was available.

In order to generate the simulations, hosts were placed at random in a bi-dimensional space. Each host was assigned a bandwidth between 15 and 70 KB/s, while latency varied from 100 to 665 ms between any pair of them. It was assumed that these magnitudes were symmetrical ($A \rightarrow B = B \leftarrow A$). Furthermore, it was considered that any host could transmit data messages with equal probability. It is important to notice that knowing the topology in advance, which, besides is static, favors the MST alternatives. Therefore this scenario is rather unfair to GMAC.

Additionally, experiments were done to evaluate GMAC with MoviLog specific group communication requirements. To handle code mobility efficiently, each MoviLog platform needs to multicast information in the following settings:

- Setting A: Each MoviLog platform needs to announce its new available resources, as well as disposing the old ones. For this kind of information, MoviLog sends, in average, two messages every 10 min. Each message has an approximated size of 2 KB.
- Setting B: Some specific MoviLog platform information must be shared periodically. This information includes MoviLog metrics and current control state information such as CPU load, number of active agents, memory usage, etc. This type of messages are sent twice in a minute and are 150 Bytes long.

In order to avoid traffic congestion, the maximum number of participants in a group is determined by the node with less bandwidth capacity. If the number of participants grows further, the node with lower bandwidth could act as a bottleneck, and congestion may occur. For example, the maximum number of members supported for the case of MoviLog, when 15 KB/s is the bandwidth of a node acting as bottleneck, is:

- Setting A : $2\text{ KB} * 2/600\text{ s} = 0.0066\text{ KB per s} = 2250/2 = 1125$ hosts.
- Setting B : $0.150\text{ KB} * 2/60\text{ s} = 0.005\text{ KB per s} = 3000/2 = 1500$ hosts.
- Both: $0.005 + 0.0066 = 0.01166\text{ KB per s} = 642$ hosts.

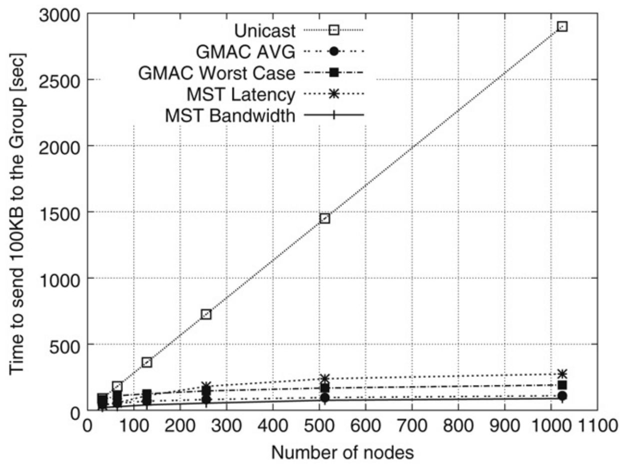


Fig. 9. Time (in seconds) to send a 100 KB message.

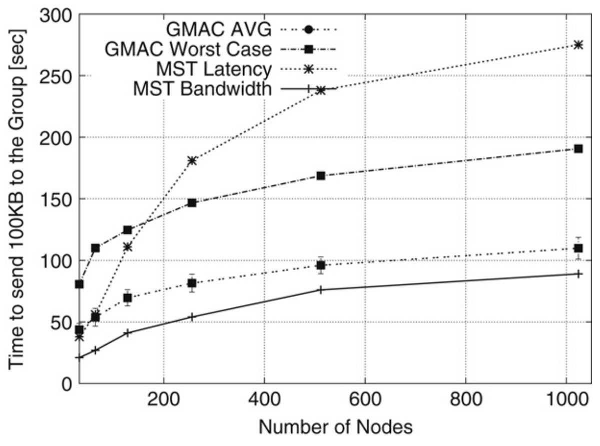


Fig. 10. Time (in seconds) to send a 100 KB message (without unicast).

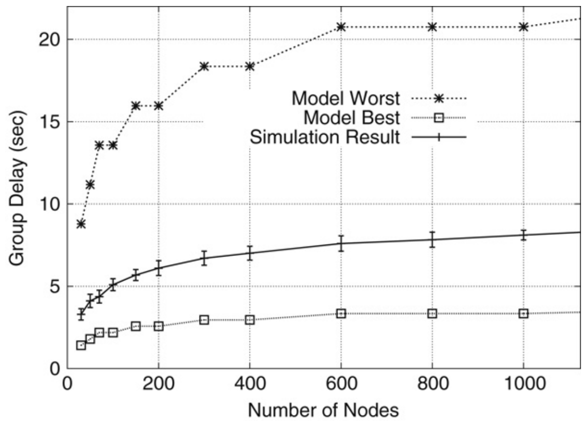


Fig. 11. 2 KB MoviLog data message delay.

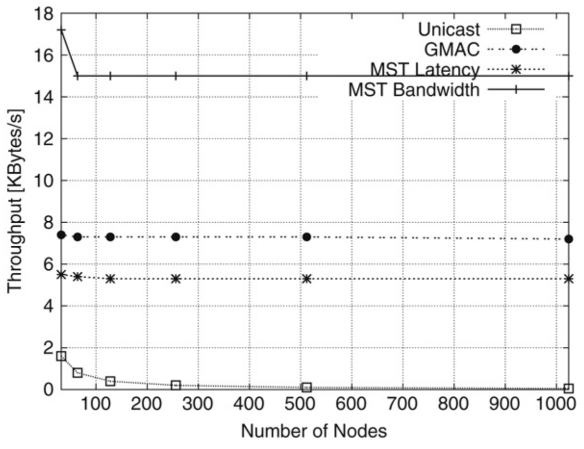


Fig. 12. Maximum group throughput.

These values provide an estimate of the feasible scalability of GMAC, where a node limited to a bandwidth of only 15 KB/s is retransmitting messages. In addition, a typical MoviLog deployment consists of approximately 400 hosts. Note that although GMAC does not currently address congestion control, the numbers shown above suggests that GMAC is adequate given its scalability requirements.

7.1. End-to-end group propagation delay

Figs. 9 and 10 show the time in seconds required by a random node to send 100 KB to all the group members. A message of 100 KB was used to have a better insight of how the node bandwidth affected the overlay tree. Results concerning MoviLog requirements will be shown subsequently. Fig. 9 shows that the alternative of using only unicast is not viable when groups are large. Fig. 10 shows in more detail the results for GMAC average and worst case, and the two MST alternatives. For the worst case, each node in the longest path is formed by bottlenecks of 15 KB/s bandwidth and 665 ms latency. The MST latency approach has poor performance as it does not take into account bandwidth metrics. The MST Bandwidth was built optimizing a 100 KB message transmission, thus this is the best that can be achieved in an overlay tree. In average GMAC scales well, approximating the boundaries imposed by the MST Bandwidth. Standard deviations of about 9% can be seen as error bars, showing that the delivery time vary only a few seconds. It is worth noting that when the number of nodes grows, they become closer to each other and the MST configuration impact tends to be less decisive. Results corresponding to MoviLog requirements are shown in Fig. 11. The time required until the last node receives a 2 KB message for MoviLog setting A is shown, both for the simulation results and the model worst and best case. For the model worst case, 15 KB/s bandwidth and 665ms lantency links were assumed in Eq. (1), whereas for the model best case, 70 KB/s and 100 ms values were used. These ranges work as over-dimensioned boundaries for the MoviLog setting. The results show how GMAC scales, since just a few seconds are required for a message to reach the whole group with a logarithmic behavior.

7.2. Throughput

Fig. 12 shows the group throughput that can be achieved with the different approaches. This metric is important for applications requiring audio/video streaming, which is not of particular interest to GMAC. In distribution trees, less capable nodes act as bottlenecks, limiting the overall group to their bandwidth. In GMAC, bottleneck nodes may have to retransmit messages, thus the overall throughput would be restricted to half of the less capable host bandwidth. It is worth noting that GMAC was developed for homogeneous groups, thus bottleneck nodes are less likely to appear. Furthermore, as mentioned earlier, GMAC is not concerned about providing media streaming. Nevertheless, in order to scale without trouble, there is a minimum bandwidth requirement in each node depending on the group transfer rate desired. For the MoviLog settings, the minimum bandwidth

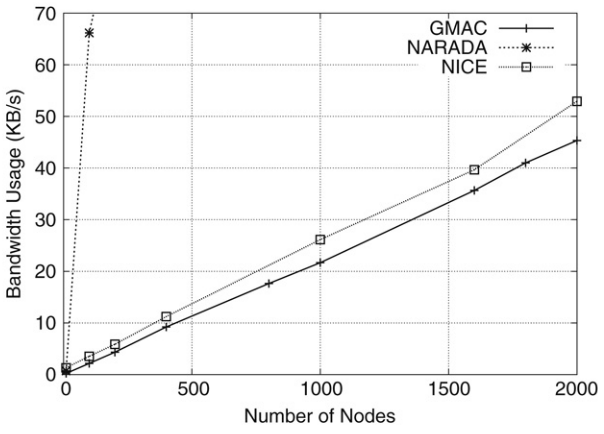


Fig. 13. MoviLog bandwidth requirements.

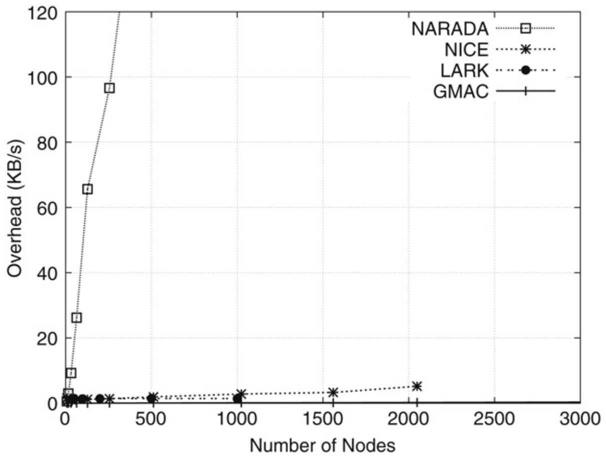


Fig. 14. Protocol overhead comparison.

Table 1
Protocol overhead (in KB/s)

Nodes	NARADA-30	NICE	LARK	GMAC
8	0.61	1.54	–	0.000818
16	2.94	0.87	–	0.001646
32	9.23	1.03	–	0.003302
64	26.20	1.20	1.42	0.006645
128	65.62	1.19	1.39	0.013190
256	96.62	1.36	1.40	0.026281
512	199.96	1.93	1.42	0.052393
1024	–	2.81	1.41	0.104827
1560	–	3.28	–	0.16154
2048	–	5.18	–	0.210454
4000	–	–	–	0.412099

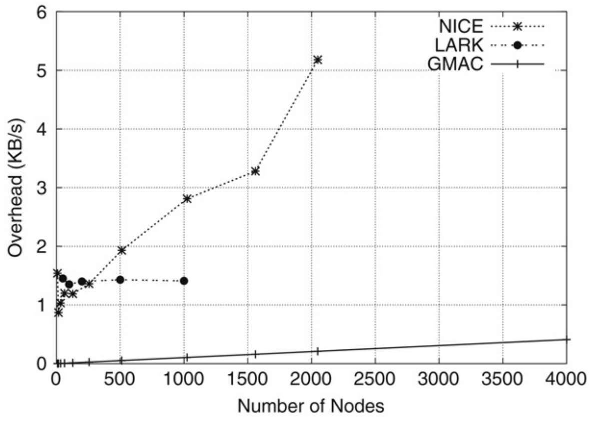


Fig. 15. Protocol overhead (without NARADA).

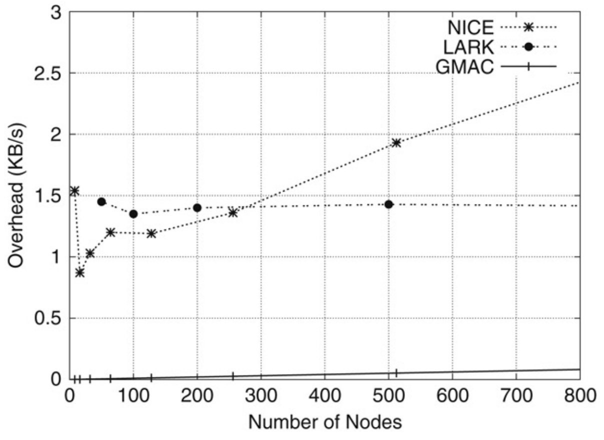


Fig. 16. Protocol overhead.

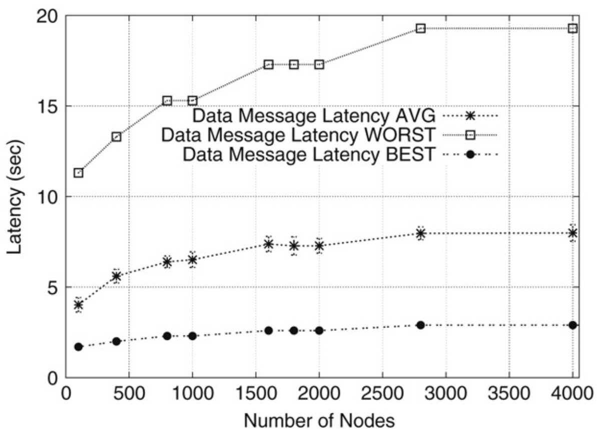
required in each node is depicted in Fig. 13. A typical MoviLog deployment with 400 hosts requires only 10 KB/s in every node in order to work properly under GMAC. NICE minimizes the control message exchanges, thus reducing the protocol overhead. Conversely, other approaches, such as NARADA, consume too much bandwidth and network resources for optimization purposes. In GMAC, scalability is not compromised, as the required bandwidth in each node is only affected by the amount of data that node is forwarding and, as detailed next, the protocol overhead is minimum and not affected by the group size.

7.3. *Protocol overhead*

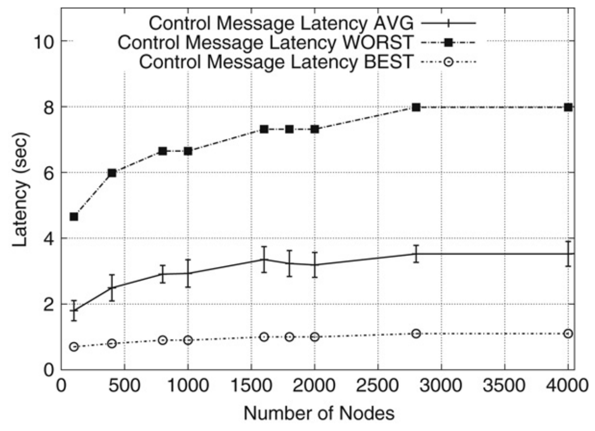
The protocol overhead metric considers network traffic that does not represent useful application data. In GMAC the information required to build and maintain the binary tree is minimal, since this information is transmitted only when a node leaves or joins the group. In contrast, alternatives using some kind of optimization introduce a considerable overhead. This is due to the fact that many measurements between hosts are required for building and maintaining the overlay structures. For instance, NARADA introduces an exponential overhead of 2% for 64 members and 4% for 128, with a single host streaming at 128 kbps [6,16]. This overhead tends to grow in large groups, confining optimized structures to small groups, or restricting them to worse approximations based on complex heuristics. Nice and LARK are application level protocols intended to reduce the exponential control overhead of NARADA.

GMAC control overhead is only affected by the failure and joining rates. The failure rates used for NICE were 16 members in 100 s in a group of 128. Although GMAC assumes that group members are stable, this failure rate was used in order to get fair comparisons. Table 1 shows the protocol overhead bandwidth produced by NARADA (with 30 s periodic refresh rate), NICE, LARK and GMAC according to different group sizes. Fig. 14 shows that NARADA exponential protocol overhead prevents it from scaling. In Figs. 16 and 15 the same metrics are shown with a different scale. It is worth mentioning that some values for larger group sizes are missing since they were not available for the alternatives compared [17,16].

While NICE and LARK reduce NARADA exponential overhead, in GMAC, as no optimization is made, the protocol overhead is minimal, and it is only affected by the overall joining or departure



(a) Data latency (group latency).



(b) Control latency.

Fig. 17. Group latency

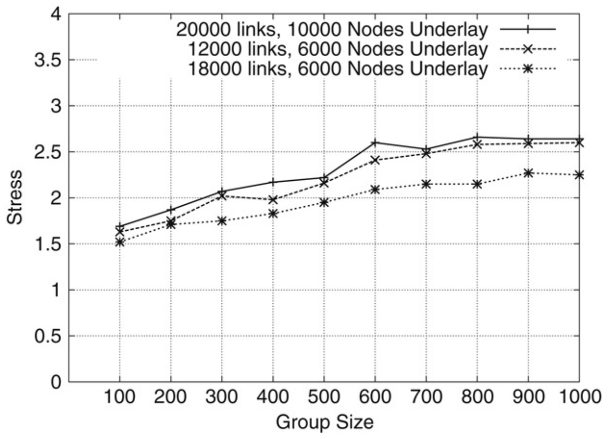


Fig. 18. Link stress.

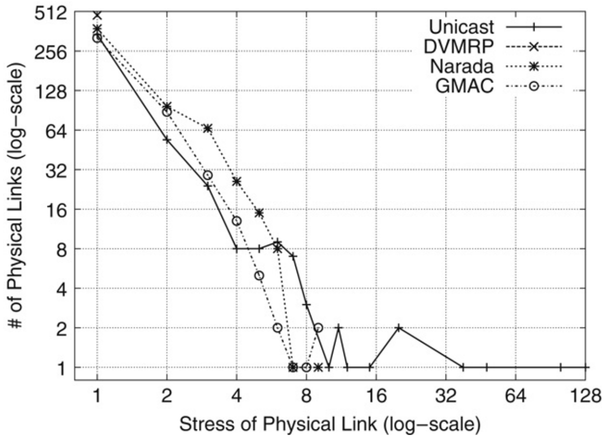


Fig. 19. Link stress distribution.

rate. In order to evaluate GMAC the failure rates proposed by the alternatives where used, where 12.5% of the nodes from a group fail within 100 s. For example for a group of 3000 nodes, 375 nodes are supposed to fail in that time. The failure rates in MoviLog platforms usually are much lower (less than 0.5% in 100 s). In addition, the protocol overhead presented in the alternatives do not include failure recovery overheads. As a consequence, GMAC probed to be the one with less protocol overhead in MoviLog settings.

7.4. Group latency

One important metric is the group latency, which is the longest end-to-end delay between any pair of nodes. Fig. 17 shows the logarithmic behavior for GMAC end-to-end group delay simulation results for data and control messages. The latency range for the underlying unicast links varied from 100 ms to 665 ms, a latency range likely to be found in the Internet. The delay was measured for the average and worst scenario (when every node in the longest path has 665 ms latency), for the following cases:

- *Data message latency* (Also known as group latency): Represents the time required to reach the farthest node from any other node in the binary tree.
- *Control message latency:* Represents the time for a control message to reach the root, since control messages flow bottom-up in GMAC.

In MoviLog it is desirable that data messages reach all members quickly. The same happens with control messages, which are sent when a node joins or leaves the overlay. As control messages flow bottom-up, in the worst case, a leaf control message would have to reach the root, generating $\log_2 n$ control messages.

It is worth mentioning that the standard deviations (shown as error bars) for the average cases are far from the best and worst case boundaries since the paths delay is counterbalanced by high and low latency links.

7.5. Network level metrics

As explained in Section 4.2, GMAC does not consider the underlying network properties when building the overlay. One of the disadvantages of this approach is that messages may be sent several times over the same physical link. Fig. 18 shows the average physical link stress for several Internet topology configurations created with the BRITE topology generator [21,22] using the waxman topology model. The link stress distribution of GMAC, NARADA, sequential unicast and DVMRP are shown in Fig. 19, reproducing the waxman generated topology (1024 nodes, and 3072) described in [6]. The stress of any physical link is at most 1 for DVMRP (Distance Vector Multicast Routing Protocol). In sequential unicast some physical links have a stress above 16, while in GMAC and NARADA the distribution is more balanced and no physical links have a stress larger that 9. This behavior is understandable, as the approach followed by GMAC is to relief the overloaded senders. From the application level perspective, as the underlying network topology information is not always available, the latency between hosts is often used as a metric of distance. The MST lantency approach minimizes the overall distance under

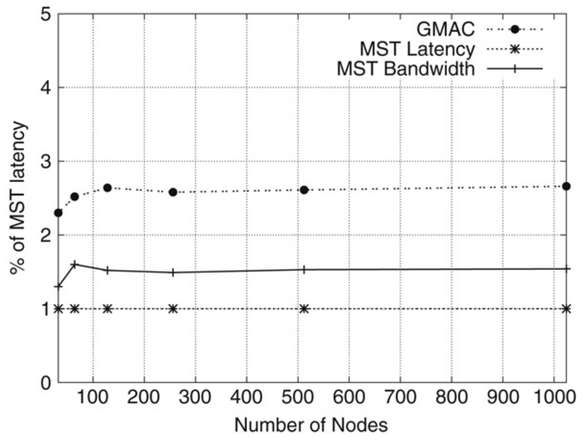


Fig. 20. Latency as network distance.

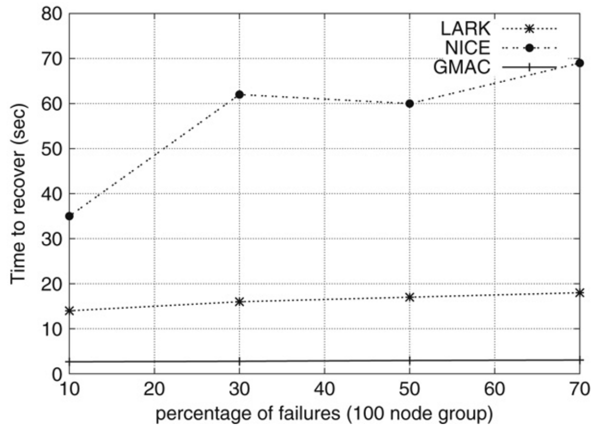


Fig. 21. Failure recovery on a 100-node group.

this assumption. The relation between the MST latency, the MST bandwidth and GMAC is shown in Fig. 20, where the results are normalized with respect to the MST lantecy. ALMI relies on this metric to build its overlay tree. NARADA, in contrast, suggests that this may not be the best choice, since the shortest path is not always the fastest one [6]. Furthermore, the assumption that on the Internet latency represents distance is not necessarily true.

To improve the resource utilization, some efforts take into account the underlying network when building overlay networks [27,19,26,35]. As explained in Section 4.2, GMAC refrains from using optimized schemes to minimize the protocol overhead, ease the overlay construction, maintenance, and failure recovery mechanisms. Other approaches such as LARK [17] also relinquishes resource utilization to increase robustness.

On the other hand, media streaming applications should be more concerned about resource utilization. As high bandwidth traffic is transmitted, it is worthwhile to afford measurements costs in order to reduce the overall resource utilization. Conversely, in GMAC, as sporadic small messages are sent, it is more important to not overload the network by exchanging metrics between nodes, since the management of an optimized structure may imply more resource utilization costs.

7.6. Failure recovery

Another important issue is the possibility of failure of some nodes. Even though the probability of a node failure may be small, when groups are large, this starts to be a significant matter. When a failure occurs in an overlay structure, members depending on the failing host, as well as the following dependents in the chain, can get disconnected. Fig. 21 shows the time required to recover from simultaneous failing hosts in a group with 100 members. As explained in Section 5.4, GMAC quickly recovers from node failures by reincorporating immediately the orphan nodes along with their subtrees in a decentralized way. In addition to the fact that the tree is not affected when a leaf fails, which, in the case of GMAC, corresponds to half of the group nodes, thus reducing the activation of the failure recovery mechanism to half the failure probability.

However, recovery in optimized overlay structures, such as ALMI, is more expensive because a host failure may trigger the reoptimization of the whole structure, producing additional protocol overhead. Moreover, complex heuristics may be required in order to keep the diffusion group working until a new optimization is performed. This is one of the issues that has hindered scalability in approaches based on optimized overlay structures. Fig. 22 shows the GMAC behavior for larger groups when 10 nodes fail. In addition to the average recovery time, the time including forcing always a root failure is shown. For the latter case, one of the orphan nodes will first replace the missing root, and afterwards receive the reconnection requests from the other nodes. In both cases GMAC quickly recovers from failures in a scalable and decentralized way.

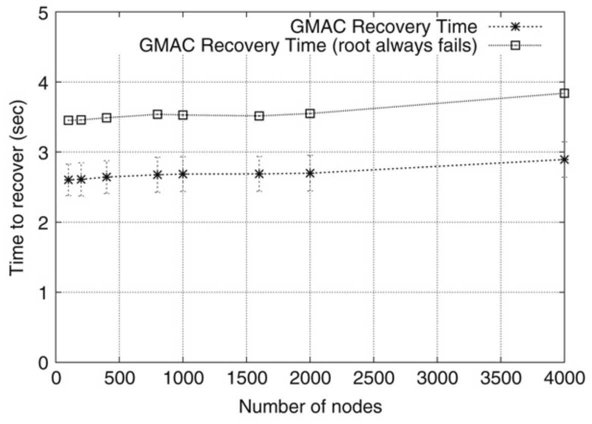


Fig. 22. GMAC failure recovery.

8. Conclusions

This paper presented GMAC, an overlay multicast network for mobile software agent platforms. These platforms reduce network usage by migrating code, thus working with the resources locally. Using a binary tree as overlay structure, GMAC allows group communication between platforms spread across the Internet. The entire functionality of GMAC is implemented in each host at the application level in a decentralized and collaborative way, thus responsibilities for message delivery, tree building and recovery are distributed among the group members.

The goal of GMAC is to provide multicast services to non-critical burst transmitting applications in a scalable and robust way, where the key idea is that overloaded hosts are relieved by delegating their transmission responsibilities to their neighbors. In addition, GMAC can be widely deployed, because it does not rely on special routers and its application program interface is compatible with the well known Java native multicast service. Furthermore, GMAC only assumes unicast as the subjacent service, allowing any host in the Internet to use it, even those with connectivity restrictions such as hosts behind NATs or firewalls.

Experimental results based on MoviLog requirements show that GMAC is a good choice for providing multicast services to groups of considerable size with small throughput transmission requirements. In contrast, approaches based on optimized structures present a number of problems such as the complexity of generating these structures, management overhead and failure recovery. In GMAC, these problems are not present, as the complexity of generating its structure is not affected by the size of the group. In addition, protocol overhead is minimal and failure recovery is fast and effective. To reduce construction and maintenance costs, GMAC does not take additional metrics for optimization purposes. Accordingly, network resource utilization is rather poor. This is one of the main drawbacks of GMAC. However this problem is also present in most overlay based approaches.

GMAC works in a fully decentralized way, building an overlay network as a binary tree. Once the tree is established, internal nodes forwards incoming messages to their neighbors. This approach is possible since the recursive nature of the binary tree allows the ascending control message heuristic. Hence a parent node receives only one control message from each child, summarizing the control state information of that entire branch. Thus the upward nodes are not overloaded by control messages. In contrast, most scalable self-organizing P2P approaches [30,28,4,32,37] do not have a defined overlay network structure that could instrument a similar control message scheme. In the near future it would be possible to design a new fully decentralized P2P overlay network based on GMAC, where queries would be multicasted instead of flooded (or other complex content based heuristic) and still behave in a logarithmic manner.

GMAC main contribution is to provide multicast services in a decentralized and robust way, where group members cooperate among each other in a fair way, minimizing the protocol overhead and thus achieving great scalability.

GMAC proposes a new way of providing application-level multicast services, therefore new possibilities of future investigation arise, as well as optimizations and improvements on the current implementation. Some of them are:

- *End-to-end Reliability:* TCP connection links in the overlay network do not ensure reliability [1]. When a host in the group fails, the data it was supposed to forward may not be delivered. The same happens on the subtrees that get temporarily disconnected. Different alternatives for message delivery could be provided in the future, depending on the applications requirements. For instance, approaches may encompass global message ordering for retransmission, token based message transmission, etc.
- *Optimization:* GMAC does not implement any kind of optimization on its structure. Nevertheless by making some node swaps it would be possible to improve the global service performance (e.g. to detect bottleneck nodes and to move them as leaves), reduce link stress, or take into account the heterogeneity of nodes and links, however, preserving the binary structure introduced by the model.
- *Group Publication Policies:* The actual implementation of GMAC allows every host to register a new group. However, this task should be done by some authorized entity, so as to gain more control over group creation and administration.
- *Security:* Some security issues must be improved. GMAC trusts hosts owning the name and password of a group. However, hosts could be tampered to behave in an inadequate way, affecting the normal behavior of the whole group.
- *Evaluation with other deployments:* we are evaluating GMAC by using two applications developed with MoviLog, a meeting scheduler multi-agent system and a workflow engine.

Acknowledgments

This research was supported by ANPCyT (Agencia Nacional de Promoción Científica y Tecnológica). We thank the anonymous reviewers for their helpful comments and suggestions to improve the quality of the paper.

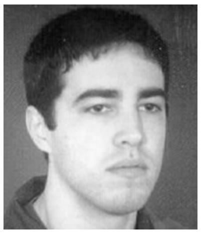
References

[1] Y. Amir, C. Danilov, Reliable communication in overlay networks, in: Proceedings of the IEEE International Conference on Dependable Systems and Networks, DSN03, IEEE Computer Society, Los Alamitos, CA, USA, 2003, pp. 511–520.
[2] S. Androutsellis-Theotokis, D. Spinellis, A survey of peer-to-peer content distribution technologies, ACM Comput. Surv. 36 (4) (2004) 335–371.
[3] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable application layer multicast, in: SIGCOMM'02: Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press, 2002, pp. 205–217.
[4] M. Castro, P. Druschel, A.-M. Kermarrec, A.I.T. Rowstron, Scribe: A large-scale and decentralized application-level multicast infrastructure, IEEE Journal on Selected Areas in Communications 20 (8) (2002) 1489–1499.
[5] Y. Chawathe, Scattercast: An adaptable broadcast distribution framework, Multimedia Systems 9 (1) (2003) 104–118.
[6] Y.-H. Chu, S.G. Rao, H. Zhang, A case for end system multicast, IEEE Journal on Selected Areas in Communication (JSAC), Networking Support for Multicast 20 (8) (special issue).
[7] C. Diot, B.N. Levine, B. Lyles, H. Kassem, D. Balensiefen, Deployment issues for the IP multicast service and architecture, IEEE Network 14 (1) (2000) 78–88.
[8] H. Eriksson, MBONE: The multicast backbone, Communications of the ACM 37 (8) (1994) 54–60.
[9] B. Ford, P. Srisuresh, D. Kegel, Peer-to-peer communication across network address translators, in: USENIX Annual Technical Conference, General Track, USENIX, 2005, pp. 179–192.
[10] P. Francis, Is the internet going NUTSS?, IEEE Internet Computing 7 (6) (2003) 94–96.
[11] P. Francis, Yoid: Extending the internet multicast architecture, Tech. rep., AT&T Center for Internet Research at ICSI (ACIRI), 2000.
[12] A. Fuggetta, G.P. Picco, G. Vigna, Understanding code mobility, IEEE Transactions on Software Engineering 24 (5) (1998) 362–375.
[13] C. Gkantsidis, M. Mihail, A. Saberi, Random walks in peer-to-peer networks: Algorithms and evaluation, in: P2P Computing Systems, vol. 63, Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 2006, pp. 241–263.
[14] P. Gotthelf, M. Mendoza, A. Zunino, C. Mateos, GMAC: An overlay multicast network for mobile agents, in: Proc. of the VI Argentine Symposium on Computing Technology, AST 2005 - 34 JAIIO, 2005, pp. 20–24.
[15] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, J.W. O'Toole Jr., Overcast: reliable multicasting with an overlay network, in: Proceedings of the Fourth USENIX Symposium on Operating Systems Design and Implementation, OSDI 2000, USENIX Assoc, Berkeley, CA, USA, 2000, pp. 197–212.
[16] S. Jin, A. Bestavros, Small-world internet topologies: Possible causes and implications on scalability of end-system multicast, Computer Networks 50 (6) (2006) 648–666.
[17] S. Kandula, J.-K. Lee, J.C. Hou, LARK: A light-weight, resilient application-level multicast protocol, in: IEEE 18th Annual Workshop on Computer Communications, CCW 2003, IEEE, 2003, pp. 201–209.
[18] D.B. Lange, M. Oshima, Seven good reasons for mobile agents, Communications of the ACM 42 (3) (1999) 88–89.
[19] Liu, Xiao, Ni, Zhang, Location-aware topology matching in P2P systems, in: INFOCOM: The Conference on Computer Communications, in: Joint Conference of the IEEE Computer and Communications Societies, vol. 4, 2004, pp. 2220–2230.
[20] C. Mateos, A. Zunino, M. Campo, Extending movilog for supporting web services, Computer Languages, Systems & Structures 31 (1) (2007) 11–31.
[21] A. Medina, A. Lakhina, I. Matta, J. Byers, Brite: An approach to universal topology generation, in: MASCOTS'01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS'01, IEEE Computer Society, Washington, DC, USA, 2001, p. 346.
[22] A. Medina, I. Matta, J. Byers, On the origin of power laws in internet topologies, SIGCOMM Comput. Commun. Rev. 30 (2) (2000) 18–28.
[23] R. Monson-Haefel, Enterprise JavaBeans, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.
[24] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, ALMI: An application level multicast infrastructure, in: Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems, USITS '01, USENIX, San Francisco, CA, USA, 2001, pp. 49–60.
[25] S. Ratnasamy, A scalable content-addressable network, Ph.D. Thesis, University of California at Berkeley, 2002.
[26] S. Ratnasamy, M. Handley, R.M. Karp, S. Shenker, Topologically-aware overlay construction and server selection, in: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Society, INFOCOM-02, in: Proceedings IEEE INFOCOM 2002, vol. 3, IEEE Computer Society, Piscataway, NJ, USA, 2002, pp. 1190–1199.

[27] S. Ren, L. Guo, S. Jiang, X. Zhang, Sat-match: A self-adaptive topology matching method to achieve low lookup latency in structured p2p overlay networks, ipdps 01 (2004) 83a.
[28] M. Ripeanu, A. Iamnitchi, I. Foster, Mapping the gnutella network, IEEE Internet Computing 6 (1) (2002) 50–57.
[29] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, STUN—Simple traversal of User Datagram Protocol (UDP) through Network Address Translators (NATs), Internet Engineering Task Force: RFC 3489, March 2003.
[30] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), in: Lecture Notes In Computer Science, vol. 2218, Springer-Verlag, 2001, pp. 329–350.
[31] R. Steinmetz, K. Wehrle (Eds.), Peer-to-peer systems and applications, in: Lecture Notes in Computer Science, vol. 3485, Springer, 2005.
[32] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, IEEE/ACM Transactions on Networking 11 (1) (2003) 17–32.
[33] I. Stoica, T.S.E. Ng, H. Zhang, REUNITE: A recursive unicast approach to multicast, in: Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications, INFOCOM-00, IEEE, Los Alamitos, 2000, pp. 1644–1653.
[34] W. Wang, H. Chang, A. Zeitoun, S. Jamin, Characterizing guarded hosts in peer-to-peer file sharing systems, in: In Proceedings of IEEE Global Communications Conference, Global Internet and Next Generation Networks, 2004, pp. 1539–1543.
[35] D. Zeinalipour-Yazti, V. Kalogeraki, Structuring topologically aware overlay networks using domain names, Computer Networks 50 (16) (2006) 3064–3082. http://dx.doi.org/10.1016/j.comnet.2005.12.003.
[36] B. Zhang, W. Wang, S. Jamin, D. Massey, L. Zhang, Universal IP multicast delivery, Computer Networks 50 (6) (2006) 781–806.
[37] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J.D. Kubiatowicz, Tapestry: A resilient global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications 22 (1) (2004) 41–53.
[38] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, J.D. Kubiatowicz, Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination, in: Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video, ACM, 2001, pp. 11–20.
[39] A. Zunino, M. Campo, C. Mateos, Simplifying mobile agent development through reactive mobility by failure, in: G. Bittencourt, G. Ramalho (Eds.), Advances in Artificial Intelligence, in: Lecture Notes in Computer Science, vol. 2507, Springer-Verlag, 2002, pp. 163–174.
[40] A. Zunino, C. Mateos, M. Campo, Reactive mobility by failure: When fail means move, information systems frontiers, Special Issue on Mobile Computing and Communications 7 (2) (2005) 141–154.

Pablo Gotthelf received his Systems Engineer degree from UNICEN, Tandil, Argentina on February 2005. He is currently a Ph.D. candidate in Computer Science focusing on P2P networks to support decentralized Virtual Organizations. His main research interests include P2P systems, multi-agents systems and collaborative applications.

Alejandro Zunino received a Ph.D. degree in Computer Science from UNICEN in 2003. He is an Adjunct Professor at the Computer Science Department since April 2006 and Assistant Researcher at the National Council for Scientific and Technological Research (CONICET) since July 2005. His research is focused on distributed computing, including Grid computing, service oriented computing and mobile agents. He has published over 30 papers in journals and conferences.



Cristian Mateos received a Ph.D. degree in Computer Science from UNICEN, Tandil, Argentina, in 2008. He is a Teaching Assistant at the Computer Science Department of UNICEN. His recent thesis was on solutions to ease Grid application development and tuning through dependency injection and policies. He is mainly interested in parallel and distributed programming, with a special emphasis on methods for gridifying applications, application-level parallelization, Grid middlewares and platforms, Service-Oriented Computing and Web Services.

Marcelo Campo received a Ph.D. degree in Computer Science from UFRGS, Porto Alegre, Brazil. He is an Associate Professor at the Computer Science Department and Head of the ISISTAN. He is also a research fellow of the CONICET. His interests include intelligent aided software engineering, software frameworks and architecture, agent technology and software visualization. He has over 70 papers published in conferences and journals about software engineering topics.