INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2012; **25**:491–514
Published online 23 May 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/dac.1275

GPM: A generic and scalable P2P model that optimizes tree depth for multicast communications

Mourad Amad[1,*,†], Ahmed Meddahi[2], Djamil Aïssani[1] and Gilles Vanwormhoudt[2]

[1]*L.A.M.O.S. (Laboratory of Modelization and Optimization of Systems), University of Bejaia, 06000 Bejaia, Algeria*
[2]*Institut Telecom/Telecom Lille 1, France*

SUMMARY

Group communications (*real-time and non-real-time*) refer to one-to-many or many-to-many communications. On the one hand, multicast is considered as an appropriate solution for supporting group communication-oriented applications (*we distinguish IP network multicast from application layer multicast*). On the other hand, peer-to-peer model tends to be a good candidate for supporting today Internet applications (*e.g. P2P IPTV, P2P VoIP, etc.*). In this context, P2P has attracted significant interest in the recent years. This is mainly due to its properties that also make P2P well adapted to today social networks. In this paper, we propose GPM (*Generic P2P Multicast*): a novel generic and scalable approach, that optimizes multicast tree depth in P2P networks (*structured and unstructured*), and contributes to control the network overlay latency. For multicast tree construction, the approach we propose is based on a distributed algorithm using a specific data structures (*adjacency and forwarding matrixes*). GPM model inherits from P2P attributes such as scalability, flexibility and fault tolerance, while taking into consideration the respective characteristics of one-to-many and many-to-many type of applications. We also give a performance evaluation for validation and comparison purposes while considering some main existing application layer multicast protocols. Copyright © 2011 John Wiley & Sons, Ltd.

Received 2 August 2010; Revised 6 January 2011; Accepted 6 March 2011

KEY WORDS: GPM; P2P; application layer multicast; multicast tree depth

1. INTRODUCTION

For one-to-many and many-to-many multimedia applications, such as video on demand, media streaming or media conferencing, the efficient and optimal distribution of the media flow to a large group of receivers constitutes a key requirement. In response to such a requirement, the efficient support of multicasting by network layer components (*i.e. routers*) was proposed in the form of network IP multicasting. However, the ubiquitous deployment of IP multicasting has been challenged by several commercial issues, as well as technical challenges related to scalability, quality of service support, security access or multicast sessions control and management [1].

The relative slow deployment of IP multicast leads to an application layer multicast approach [2]. Application layer multicast refers to the implementation of multicast capability at the application layer (*end hosts*) instead of network layer (e.g. *building a multicast-capable overlay network over a unicast-infrastructure*). This approach provides relative benefits compared to network IP multicast.

*Correspondence to: Mourad Amad, Laboratory of Modelization and Optimization of Systems, University of Bejaia, Targa Ouzemour, 06000 Bejaia, Algeria.
†E-mail: mourad.amad@univ-bejaia.dz

They can be used to overcome deployment barriers to router-level solutions of several networking problems, they also offer flexibility, adaptivity and ease of deployment [3].

Recently, application layer multicast approaches have been applied to the P2P domain for providing some key benefits [4, 5] such as self-organization, scalability, fault tolerance and robustness. Multicast in P2P enables applications such as P2P IPTV and more generally P2P group communications [1]. In multicast P2P system with high membership turnover (*usual referred as churn*), any node can be the source of a data flow for potentially a large number of receiver nodes. This causes challenging issues when designing a multicast P2P mechanism. One key issue is to distribute the media flow with efficiency between multiple and independent participants or conference groups. Another key issue is to deal with the dynamic and potentially high churn rate during the media flow distribution.

In this work, we propose GPM (Generic P2P Multicast), a novel approach that aims to deal with both these issues. The main characteristics of GPM are decentralization (*with greedy decision for each node*), operability (*in terms of implementation cost*) and scalability. In terms of efficiency, GPM relies on multicast tree and aims to optimize the method for building the multicast tree from any node (*source*) to the other receiver nodes in a P2P network. The optimization consists in minimizing the depth of multicast tree (*in terms of hops*) and consequently the overlay end-to-end delay. It is based on a specific data structure called `adjacency matrix` that is built at each node from their finger table. GPM is also able to support dynamic node or peers with a rapid convergence (*churn rate*), while join and leave operations are considered with a limited cost.

Unlike most of the existing application layer multicast solutions, GPM is not limited to one-to-many type of applications. It is also defined for many-to-many applications such as P2P multi-party conferencing. This is made possible by extending GPM with a specific data structure called `forwarding matrix` that is built during the multicast tree construction process. Thanks to this data structure, each node within a multicast tree can forward data in an optimized way and thus minimizes the overlay network traffic.

The related works on ALM (*Application Layer Multicast*) for P2P systems (e.g. [6–12]) show an intense activity. However, as opposed to the existing approaches that consider a specific architecture, the proposed GPM approach is generic regarding the underlying networks. Thus, GPM can be implemented on top of any P2P architecture (e.g. *CAN, gnutella, Chord, etc.*). In contrast with others works, we will be able to provide the results for multicast tree on both architectures and give comparison elements.

The paper is organized as follows: Section 2 gives a brief overview of P2P systems and describes the concepts of application layer multicast and network IP multicast. We also resume the main related works on application layer multicast (*ALM in P2P systems*). The proposed GPM approach (*A Generic and Scalable P2P Model that optimizes tree depth for Multicast communications*) is described in Section 3. We particularly focus on the multicast tree calculation, and describe our proposed approach for both structured and unstructured P2P overlay. In Section 4, we provide an extention of GPM for many-to-many type of applications. Section 5 illustrates the performance evaluation for GPM provides analytical and simulation results. Finally, we conclude and give some perspectives in terms of potential application scenarios and extra functionalities.

2. BACKGROUND AND RELATED WORKS

Peer-to-peer (P2P) network and application layer multicast are two pillars of the proposed GPM. They are also two alternative models, respectively, to client–server and IP multicast models. This section gives a brief overview of P2P concepts and a deep analysis of the existing P2P ALM.

2.1. Peer-to-peer networks (P2P)

Unlike client–server architecture, peer-to-peer computing allows mutual exchange of information and services directly between a sender and a receiver (*one or multiple*). It is characterized by self-organization, scalability and resilience. P2P networks support different mechanisms, mainly to

DOI: 10.1002/dac

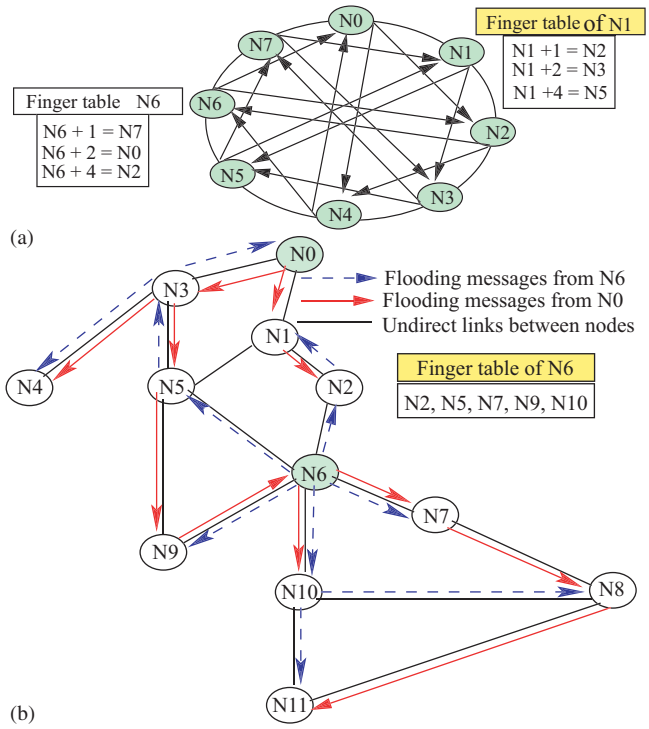

Figure 1. Example of (a) structured and (b) unstructured P2P networks.

discover, query other peers and locate resources for content distribution or sharing. P2P architectures are grouped into two main categories:

- Unstructured P2P systems (e.g. *Gnutella* [13], *Napster* [14], *ABC* [15] *and CBT* [16]) are generally based on a global index (*partially centralized*), or use a flooding algorithm to locate and discover other peers or resources [17–19]. The architecture and maintenance are globally simplified, while the scalability represents a key issue.
- Structured P2P systems (e.g. *P4L* [20], *Cycloid* [21], *Chord* [22] *and EZSearch* [23]) are based on the concept of Distributed Hash Table (DHT). With the DHT approach, each entity name in the system can be mapped into a single search space (*identifier*), by using a hash function such as SHA-1 or SHA-2. Thus, all the entities in the system have a consistent view of that mapping. Given that consistent view, various structures of the search space are defined for locating the target entities. As an example, in Chord, the search space is based on a ring topology. In P4L, it is structured on hierarchical rings.

Figure 1(a) illustrates a Chord architecture (*as a structured P2P network*) where lookup is based on DHT with a ring topology. Figure 1(b) illustrates a Gnutella architecture (*as unstructured P2P network*) where lookup is based on flooding technique with random topology. In the following, we use these two representative examples for showing the generic aspect of GPM as well as for presenting its functional principle and its performance evaluation. We give two approaches for multicast tree construction: a primitive one (*basic approach based on an epidemic technique*) that lets us to construct a multicast tree from any P2P overlay, and show that is not optimized (*especially in terms of tree depth and overlay latency*); then we introduce the optimized approach proposed by GPM that constructs a multicast tree with efficiency and optimization.

The next sub-section introduces the multicast concept by distinguishing IP multicast from application layer multicast. We also discuss and compare the existing works related to ALM for P2P networks.

DOI: 10.1002/dac

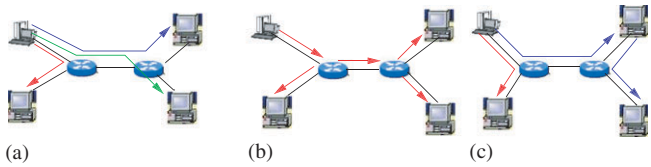

Figure 2. (a) Unicast; (b) IP multicast; and (c) application layer multicast.

2.2. *Application layer multicast* (*ALM*)

Since IP multicast has not been widely supported by major commercial Internet Service Providers (ISPs) [24], application layer multicast has been introduced. ALM aims to address unicast scalability issue by distributing data duplication process among the different group members, in an adaptative and efficient way. However, ALM is not efficient as IP multicast in terms of data replication. Given this, ALM optimization is a critical issue. Figure 2 illustrates the dissimilarities between unicast, IP multicast and application layer multicast. In unicast (Figure 2(a)), paths are constructed from a source node to each receiver. In IP multicast (Figure 2(b)), a router with multicast capabilities is needed to duplicate and send messages to each receiver. However, in ALM (Figure 2(c)), any node can be a multicast router.

ALM does not require any kind of multicast support (*from a network point of view*) as it is only based on unicast communications. However, it suffers from a lack of standardization (*not generic*). In ALM, the nodes are self-organized, based on mesh or tree topology. The mesh architecture generates high overhead and scalability issue; the tree-based architecture (*where network is initially a tree*) is not fault tolerant.

The main types of the existing protocols for implementing multicast are any source multicast (ASM), single source multicast (SSM) and application layer multicast (ALM), they are resumed as follows [25]:

- *Any source multicast* (*ASM*): ASM offers several service models that can be used to build a range of applications. However, due to significant implementation and practical issues, ASM deployment is relatively limited.
- *Single source multicast* (*SSM*): To address a number of implementation and deployment issues, SSM was introduced. Nevertheless, SSM service model considers one unique sender per session. So, applications that require multiple senders must either use different protocols or implement extra functionalities on top of SSM.
- *Application layer multicast* (*ALM*): ALM provides a solution to deployment issues of ASM and SSM, by shifting multicast forwarding process from the network to the terminals. As a result, ALM offers more flexibility than ASM and SSM. However, it is less efficient, less powerful and suffers from scalability issues.

GPM belongs to ALM classification where optimization in terms of traffic overhead, tree depth and consequently end-to-end overlay delay from a source to each receiver represents an open issue and challenge. Our contribution aims to give a scalable model for ALM, while taking into consideration the overlay network characteristics (e.g. *end-to-end overlay delay*). GPM is specifically designed for P2P architectures. Prior to its description, we give a review of related works in the following.

2.3. *ALM for P2P*

A taxonomy of application layer multicast solusions was proposed in [25]. In their paper, the authors present two classes of ALM. The first class is based on centralized algorithms, such as ALMI protocol [7]. This protocol uses a session controller node that gathers distance information from all groups of nodes, and calculates an overlay tree used by each node to discover its close neighborhood. The second class of ALM is based on distributed algorithms, which is divided into three sub-classes: (a) mesh first algorithm such as Narada [26], (b) Tree first algorithm such as NICE [27] or HMTP [6] and (c) coordinate system such as SCRIBE [16].

DOI: 10.1002/dac

Table I. GPM vs some other application layer multicast protocols.

	Narada	Nice	Scribe	Overcast	MCAN	GPM
One-to-many (1)/ many-to-many (2) type of applications	(1)	(1)	(1)	(1)	(1)	(1) (2)
Underlying topology	Tree	Hierarchical clustering architecture	Pastry	Tree	CAN	Generic
Presence of a central entities	No	Yes	No	No	No	No
Mechanism	Spanning tree	Spanning tree	Spanning tree	Spanning tree	Epidemic	Epidemic and spanning tree

In Narada [26], the overlay network is built according to a two-step process: First, for establishing and optimizing a well-connected and controlled mesh topology. Second, for building a source tree from each potential sender to every receiver using a subset of the existing mesh links. Narada protocol keeps state about all other members that are part of the group. This information is also periodically refreshed. Distribution of such state information about each member to all other members leads to a relatively high control overhead.

As opposed to Narada, ALMI [7] relies on a central session controller node to calculate a bi-directional and minimal spanning tree data distribution overlay, between the registered nodes. The session controller can be implemented on one of the participating nodes, or on a well-known external node.

Overcast [28] is a self-organized distribution tree, where nodes select appropriate parent. Overcast builds sender-specific trees instead of a single shared tree. NICE [27] uses a hierarchical clustering technique to build an overlay tree, whereby group members arrange themselves into clusters with the neighboring nodes. However, clustering-based techniques face the scalability problem, and suppose a certain complexity on supernodes (e.g. *management*). HMTP [6] builds an overlay tree by choosing receiver nodes based on a recursive selection of better parents to connect to, in a distributed way. HMTP uses a limited scope approach, because at each step of the recursive process, a node measures its distance only from nodes considered as 'children' of its 'current' parent.

Application-level protocol based on CAN [12] splits a multi-dimensional virtual torus into adjacent regions; and uses a broadcast mechanism to flood data to all regions. However, flooding techniques (*epidemic technique*) are not cost effective, particularly in terms of traffic overhead when nodes are widely dispersed. Finally, SCRIBE [11] exploits P2P characteristics, to build application-level multicast tree, by merging P2P search paths to form a tree.

In this work, we focus on combining application layer multicast with any existing P2P architectures, for inheriting and providing some key advantages such as self-organization, scalability, fault tolerance and robustness. The GPM model constructs an efficient and optimized multicast tree from any node (*source*) to the other nodes in a P2P network (*structured or unstructured*). Table I presents GPM characteristics for comparison with some existing application layer protocols discussed above. From this table, we can see that all other works only focus on one-to-many type of applications, while GPM supports both one-to-many and many-to-many type of applications. A second unique characteristics of GPM is its genericity regarding the underlying overlay networks. While others are based on a specific topology such as Tree, CAN or Pastry, GPM can be deployed on any P2P architectures. As opposed to ALMI and Nice which relies on a central entity (*super node*), GPM is distributed (*such as Narada* and *Scribe*). However, compared to other distributed approaches, GPM does not build a shared spanning tree but a source-oriented tree: each node can be a source of flow and constructs its proper multicast tree. Finally, GPM differs from the existing approaches by combining both an epidemic mechanism but only on a restricted subset of the network and a spanning tree mechanism for the rest. This combination allows to take benefit from the robustness of epidemic techniques and from the efficient coordination provided by spanning tree.

DOI: 10.1002/dac

3. GPM: PRINCIPLES AND CONCEPTS

P2P network can be represented as a direct graph $G=(V, E)$, where V and E denote, respectively, the set of network nodes and the logical directed links. From a logical topology perspective, the oriented edge from node Ni to node Nj in G represents the unicast path from Ni to Nj. Then the problem can be formulated as follows: Given any node in a P2P network, how to build an optimized tree that connects a node N (*source*) to all other nodes in the network with efficiency? Or, from any node, how to send a data flow with efficiency to a group of participants in the network? Also, how to build simultaneously several optimized multicast trees from each sender to their respective receivers in the context of many-to-many type of applications?

Given this, we propose GPM, a generic and scalable model for one-to-many type of applications, and then an extension to many-to-many type of applications is given. In the first type of applications, we construct a multicast tree from any source node to each receiver (*single source with multiple receivers*). In the second type, GPM is applied to construct a multicast tree from each source node to each receiver (*multiple sources with multiple receivers*).

3.1. Multicast tree construction process in GPM

Let us consider the multicast tree construction process from any existing P2P network between one defined node (*source*) and a subset or all other nodes (*receivers*). For this, we consider Chord [22] as one reference for structured P2P overlay; based on a ring topology (Figure 1(a)), and Gnutella (Figure 1(b)) for unstructured P2P overlay with random topology. We also consider two types of solutions for multicast tree construction: The first one based on forwarding messages inside all the current node neighborhoods (*epidemic technique such as MCAN*). This basic approach leads effectively to the construction of a multicast tree while minimizing the complexity, but is not necessarily optimized in term of overlay latency. The basic approach is introduced for illustration purpose (*we show that the problem in this approach is not the multicast tree construction by itself, but a multicast tree that is efficient, optimized and converges rapidly*).

3.1.1. Basic approach. Let us consider a basic approach based on the epidemic technique, for sending (*from a source node N*) and forwarding data (*from a relay node*) to other receiver peers. The peer N builds a multicast tree (N *is a source*) by proceeding as follows: First, node N sends a message *Child* (N) to all its neighbors (*Invite request*) meaning 'be a child of Ni'. If Child M (*successor of* N) accepts the received request, it gives a positive response (ACK) to N and forwards this request (*message Child* (M)) to all its respective neighbors, and so on.

In a P2P overlay network, a node N can have more than one directed link (*from parents to node* N). With the basic approach as described above, if the node N accepts more than one similar request messages, it will be a child for more than two nodes (Figure 3), and consequently the multicast tree construction will be corrupted (*loop*). As a solution, a node that receives and acknowledges a message *Child* (N), will not accept any other similar requests from other nodes by sending back a negative response (NACK).

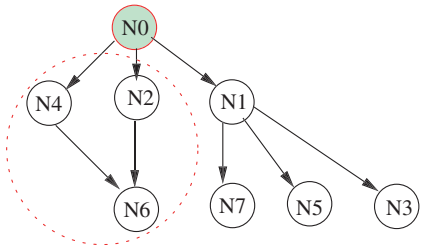

Figure 3. Corrupted multicast tree.

DOI: 10.1002/dac

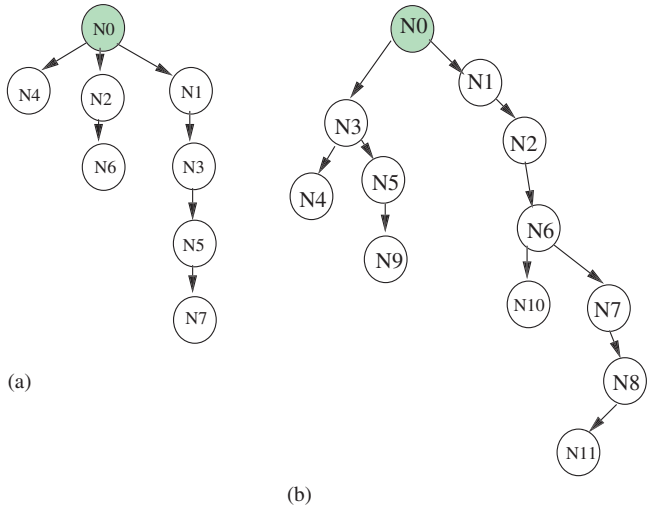

Figure 4. Multicast tree generated with the basic approach: (a) Chord and (b) Gnutella architectures.

Algorithm 1 describes the multicast tree construction process when a basic approach is considered.

Algorithm 1 : Multicast tree construction algorithm: basic approach

1: Begin
2: If (N_i *is a source node*) **then**
3: Send request **Child** (N_i) to all nodes in the finger table
4: else // N_i *is not a source* (*Relay node*)
5: At the reception of a request **Child** (N_j) **do**
6: If (N_i *has received and accepted similar request previously*) **then**
6.1: Send (NACK) and discard this request
7: Else
8: Accept this request by sending (ACK) and forwarding **Child** (N_j) to all neighbors in the finger table
9: End.

As an example, Figure 4(a) and (b) shows a multicast trees with source N0, generated, respectively, from Chord (Figure 1(a)) and Gnutella (Figure 1(b)) architectures.

Basic approach can be sufficient to construct a multicast tree from a source to multiple receivers as shown in Figure 4. Nevertheless, the constructed tree is not necessarily optimized (e.g. *in terms of overlay delay*), and leads to increase in the global overhead. From the example illustrated in Figure 1(a), the node N1 sends request *Child* (*N1*) to all its neighboring nodes (*N2, N3, N5*), and when the node N3 receives this request (*Child* (*N1*)), it forwards the request *Child* (*N3*) to its neighbors (N5, etc.). If node N5 receives this request before N1 (i.e.: *Child* (*N1*)), due to the overlay topology, it will accept the first request, while the second will be rejected, then the generated multicast tree will have a higher depth. This scenario shows that this approach is not optimized and scalability is not guaranteed (*tree depth is not controlled*).

To improve the multicast tree construction in terms of tree depth, we propose another approach that optimizes the multicast tree in terms of tree depth, which has a significant impact on the end-to-end overlay delay (*if the tree depth is minimized, the average end-to-end delay from sender to receivers is also minimized*). This approach is based on a specific data structure (*adjacency matrix*) for the multicast tree construction.

DOI: 10.1002/dac

Table II. Example of adjacency matrix for Chord network with eight nodes (Figure 1(a)).

	N0	N1	N2	N3	N4	N5	N6	N7
N0	—	1	1	0	1	0	0	0
N1	0	—	1	1	0	1	0	0
N2	0	0	—	1	1	0	1	0
N3	0	0	0	—	1	1	0	1
N4	1	0	0	0	—	1	1	0
N5	0	1	0	0	0	—	1	1
N6	1	0	1	0	0	0	—	1
N7	1	1	0	1	0	0	0	—

3.1.2. Optimized approach. The optimized approach is based on the adjacency matrix (*see* Table II), combined with a distributed algorithm (see Algorithm 3) for multicast tree construction. These two concepts are presented below.

Adjacency matrix description and construction: Based on each P2P neighbor node from the underlying P2P architecture, we define the adjacency matrix (*denoted Adj_Mat*) as follows: Adj_Mat$[i, j]=1$, if there is a direct link from node N_i to N_j (*not bijective*). Otherwise, it is equal to zero. As an example, Table II represents the adjacency matrix that corresponds to the network illustrated in Figure 1(a). On structured P2P overlay networks such as Chord, the adjacency matrix is characterized by $\sum_j \text{Mat\_adj}[i, j]=\log_2(n), \forall i$ and $\sum_i \text{Mat\_adj}[i, j]=\log_2(n), \forall j$, where n is the number of nodes in the network. The adjacency matrix is represented as a matrix of bit shared by all the nodes (*low cost in terms of memory space*), so it is cost effective in terms of resources usage; particularly, for terminals with limited capabilities (*such as PDA and mobile phone*).

The adjacency matrix represents the global knowledge at any node of all active participants; it reflects and gathers the finger table of all network nodes. Adj_Mat is expected to be identical at each node before GPM builds the multicast tree. This requirement is satisfied by Algorithm 2 as follows: Each node sends its finger table (*entry on its local adjacency matrix*) to its successors (Figure 5(a)), but also at the reception of a finger table entries from its predecessors, it updates the adjacency matrix locally and forwards the received entries (*lines*) to its successors (Figure 5(b)). Figure 5 illustrates the step-by-step process for adjacency matrix construction corresponding to Algorithm 2. In a network with eight nodes, three iterations are needed for constructing and sharing this adjacency matrix. More generally, $O(\log_2(n))$ iterations is needed for a network with n nodes.

Algorithm 2 : Adjacency matrix construction algorithm

1: Begin
2: Initialization of the adjacency matrix with a new entry corresponding to the local finger table
3: Send this entry to all successors in the finger table
4: For any changes in the finger table (*neighboring*) or at the reception of a new entry in the finger table **do**
  4.1: Update this entry in the adjacency matrix.
  4.2: Send this entry to all successors in the finger table.
5: End.

Prior to the description of the multicast tree construction process, we assume the function `level(`N_i`)` that gives the level of any node N_i (*source node is at level 0*), and then we define sets `A`, `B` and `C` as follows:

$$A = \text{set}A(N_i)=\{N_j|\text{level}(N_j)<\text{level}(N_i)\} \quad (1)$$

$$B = \text{set}B(N_i)=\{N_j|\text{level}(N_i)=\text{level}(N_j)\} \quad (2)$$

$$C = \text{set}C(N_i)=\{N_j|\exists t<i, N_t \in \text{set}B(N_i), N_t \in \text{set}A(N_j)\} \quad (3)$$

DOI: 10.1002/dac

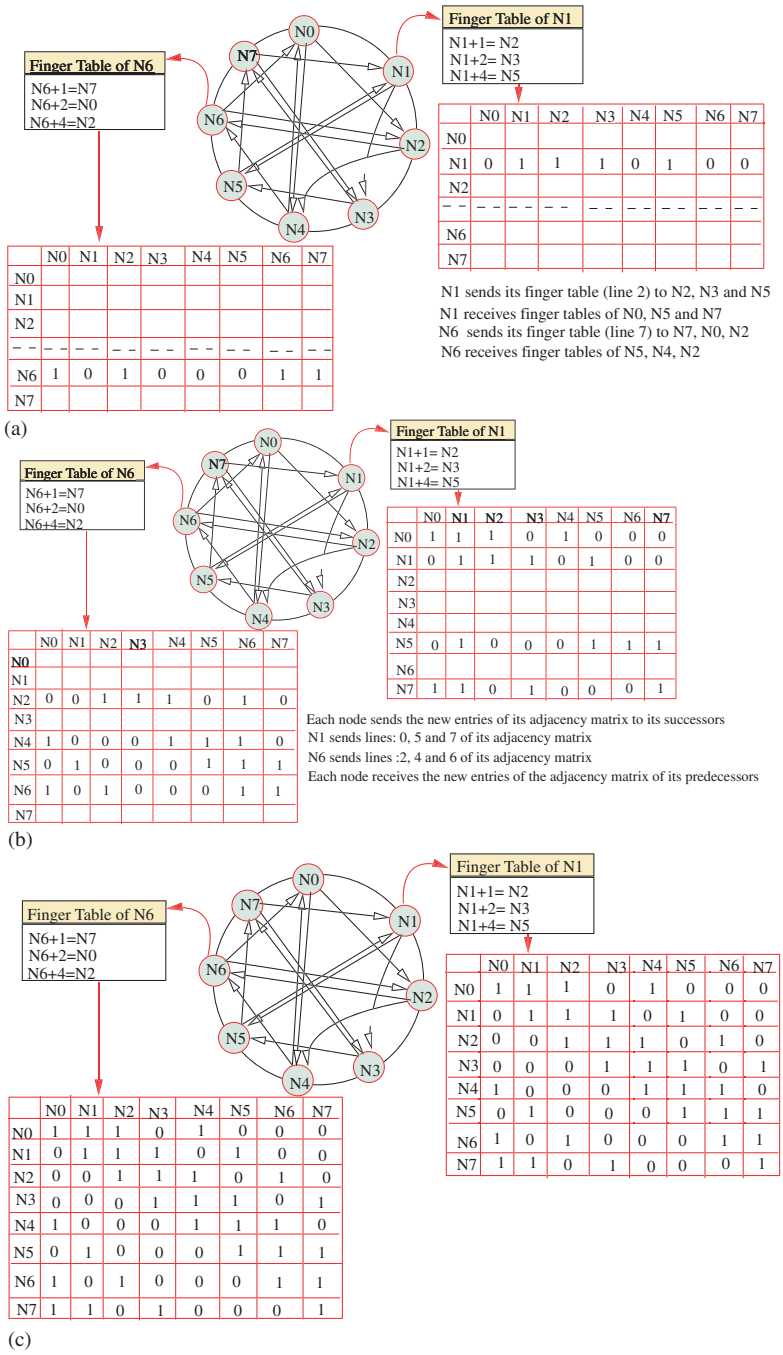

Figure 5. Adjacency matrix step-by-step construction: (a) Iteration 1: Adjacency matrix initialization; (b) Iteration 2: Adjacency matrix construction; and (c) Iteration 3: Adjacency matrix stabilization.

Algorithm 3 describes the process for building an optimized tree when N_i is a source. N_i sends message Child(N_i) to all its successors in its finger table, and at the reception of similar request (*message*) from another node, the requested node forwards the message also to all its neighboring nodes except those in sets A, B or C, and so on. These sets allow a greedy decision of GPM for the multicast tree construction. Set A allows nodes to discard a child that belongs to the higher level on the multicast tree; set B allows nodes to discard a child at the same level on the multicast

DOI: 10.1002/dac

tree, and the set C allows nodes to discard a child that are child of a node on the same level of N_i but with low identifier. When two nodes at the same level computes their related part of the multicast tree, it may occur that they have the same child, generating a conflict. To ensure the consistency of the tree, only one child must be retained. This is ensured by the ($k<i$) condition (*see example hereafter*). The sets A, B and C are derived from adjacency matrix during the multicast tree construction process (Algorithm 3) and they are stored locally at each node for each multicast session.

Algorithm 3 : Multicast tree construction algorithm: optimized approach

1: Begin
2: If (N_i *is a source node*) **then**
3: Send request **Child** (N_i) to all nodes in the finger table
4: else // N_i *is not a source*
5: At the reception of the request **Child** (N_j) from node N_j **do**
6: Forwards this request to all nodes in its finger table except those in sets **A**, **B** or **C**
where:
A: The sets of nodes which precede N_i
B: The sets of nodes at the same level of N_i
C: The set of nodes which are children of node N_k, where $N_k \in \text{setB}(N_i)$ and $k < i$
7: End.

Example

Figure 6 shows the multicast tree construction process from Chord illustrated in Figure 1(a). The node N6 (*source*) initiates the construction of the multicast tree using Algorithm 3 (*line 2*). It sends request *Child* (*N6*) to all its successors (*relay nodes: N0, N2 and N7*). Similarly, those last nodes (*relay nodes*) execute the same algorithm (*line 4*), and so on. When the node N1 receives the request (*message*) *Child* (*N0*), it accepts and forwards the request *Child* (*N1*) to all its successors except to the node N2; as it is in set A of N1 (*at a higher level*), and to the node N3; as it is in set B of N1 (*at the same level*). When the node N4 (*gray node*) receives and accepts request *Child* (*N0*) from node N0, it does not forward it to node N5 since this last node is a child of N4's brother (N1) with an identifier lower (1<4) than N4 identifier (*set C*). The result is an optimized multicast tree in terms of depth represented by the gray nodes as shown in Figure 6. We note that there are no other 'better' paths (*in terms of tree depth*) from any source to any receivers, also the end-to-end overlay latency is controlled and limited (see Section 5.1).

The generated tree is illustrated in Figure 7(a) (*resp.* Figure 7(b)) for Chord scenario (*resp. Gnutella scenario*). The tree depth in the first case is 3 when a network of eight nodes is considered (*generally* $O(\log_2(n))$), thus providing a more efficient tree construction than the basic approach (*see* Figure 4(a) and 4(b)).

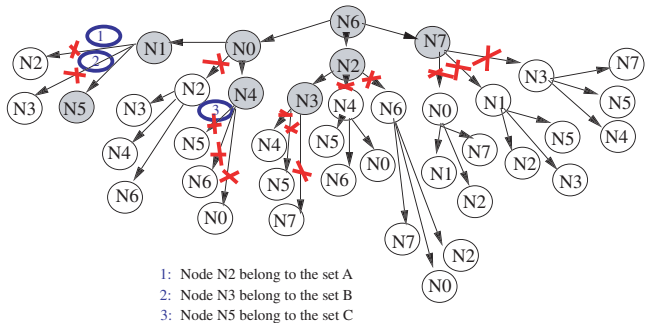

Figure 6. Example of multicast tree construction in Chord based GPM with eigth nodes.

DOI: 10.1002/dac

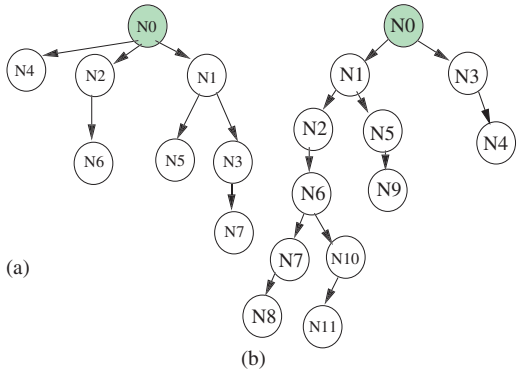

Figure 7. Multicast tree generated with the optimized approach for Chord (a) and Gnutella architectures (b).

From Figures 4 and 7, we can note that the tree depth in GPM is optimized by a ratio of 4/3 (*for Chord architecture with eight nodes*) and 6/5 (*for Gnutella architecture with 12 nodes*) comparatively to the basic approach. As we will see in the performance evaluation section, the ratio increases when the number of nodes in the network increases.

3.2. Join and leave processes

In a dynamic environment, nodes can join or leave the network at any time (*churn rate*). Given this, a bootstrapping mechanism constitutes a required key functionality for each P2P networks. Nodes intending to participate in such overlay network, initially have to locate at least one node already member of this network.

Four types of bootstrapping mechanism [29] exist: static overlay nodes-bootstrapping servers, out-of-band address caches, random address probing and employing network layer mechanism. Owing to the generic approach of GPM, the bootstrapping mechanism depends on the underlying P2P network, on which GPM is implemented.

Join operation: When a new node joins the system by sending a request `join` to the node N given by the bootstrapping mechanism, it takes place on the P2P overlay architecture. This node constructs its finger table and then the adjacency matrix. The contacted node N compares its level in the multicasts tree with the levels of their neighboring nodes (*in the overlay network*) and sends the identifier of a neighborhood node at a lower level in the tree (closer to the source). Consequently, just a portion of the multicast tree will be impacted by the update process. As shown in Figure 8, when node $N12$ joins the network, and the bootstrapping node gives $N9$ as relay node. $N12$ contacts $N9$, *N9* sends the identifier of *N5* (*neighboring at the higher level*) to $N12$, *N12* will be linked to node *N5* and receives data from it. Finger tables will also be updated by the underlying P2P stabilization algorithm. Furthermore, the adjacency matrix will be updated using Algorithm 2 of GPM.

Leave operation: When a node leaves the GPM system in the P2P architecture, a stabilization algorithm is invoked. In this case also, only a part of the multicast tree will be updated and impacted. Figure 9 illustrates the leave process for a node on a P2P. As an example, while node N10 leaves the P2P network (Figure 9(a)), this will be detected by node N11 (*keep alive or time out mechanisms*) which reconnects to another node as follows: It contacts its neighboring nodes in the network (N8), N8 sends its neighboring node at a higher level on the tree (N7) to N11. Then N11 connects to N7 and receives data flow from it. Even with join and leave operations, the generated multicast tree is kept optimized in terms of depth, after the update process.

Most of the existing application layer multicast solutions are defined for one-to-many, and some others for many-to-many applications (e.g. [30–32]). GPM could be applied for both one-to-many and many-to-many applications. The section below describes the extension of GPM to many-to-many type of applications.

DOI: 10.1002/dac

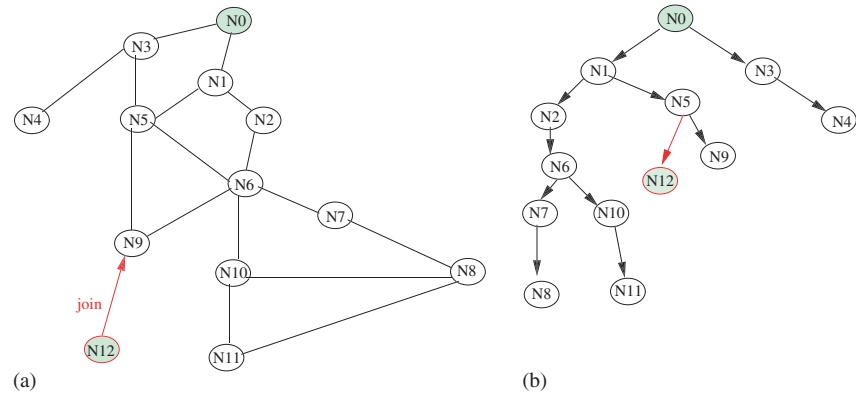

Figure 8. Joining the multicast tree in GPM: (a) N12 joins the Gnutella network and (b) multicast tree updated.

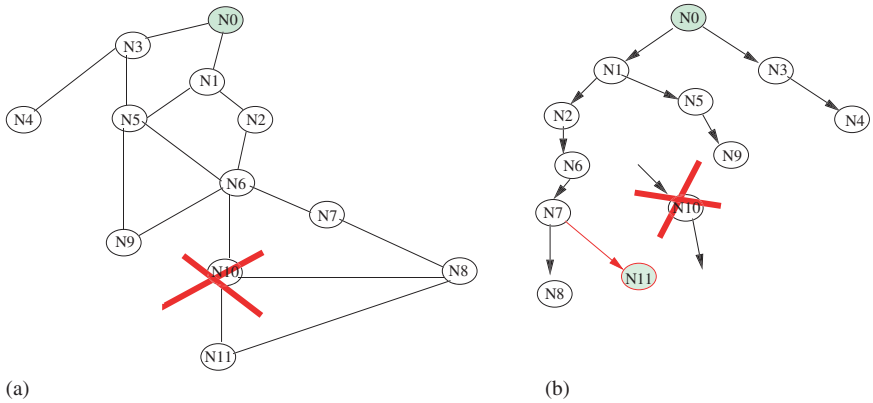

Figure 9. Leaving the multicast tree in GPM: (a) N10 leaves the Gnutella network and (b) multicast tree updated.

4. EXTENSION TO MULTI TREE FOR MANY-TO-MANY TYPE OF APPLICATIONS

For one-to-many applications, such as P2P IPTV [33], a single and optimized tree from the source (*media server*) to all other nodes is sufficient (*one source with multiple receivers*). Nevertheless, for many-to-many type of applications (e.g. *P2P multiparty conferencing*), each node (*source*) builds a source-based tree for media forwarding. Then, it is necessary to built m trees (*m sources*) in n-nodes network with $m \leqslant n$. Each tree has one source and multiple receivers, and the nodes can participate to more than one tree. A node N_i that receives data from another node N_j, should forward it using the appropriate tree (*same data flow*). This is one of the key issue for many-to-many applications.

In order to extend our proposed GPM model for many-to-many applications, we propose the following approach:

We define a new data structure called `forwarding matrix`, and denoted `Tab` (*see* Table III). Each node N builds and stores its own forwarding matrix that connects each sender node (P *immediate predecessors of* N) to its associated receiver nodes (S *immediate successors of* N). The forwarding matrix for node N is Tab$[i,k]$, where $1 \leqslant i \leqslant$ Num represents nodes from which N can receive flow, and $1 \leqslant k \leqslant$ Num represents nodes to which N sends flow. Num is the degree of the underlying P2P overlay, generally, it is equal to $\log_2(n)$ such as in Chord. The first entry of Table III contains the nodes from which N can receive data. The other entries contain the nodes to which N forwards data. The forwarding matrix is derived from Algorithm 3. As an example, the forwarding matrix of node N3 in Chord-based GPM (*see* Figure 10) is illustrated in Table III.

DOI: 10.1002/dac

Table III. Forwarding matrix for node N3 in a Chord-based GPM with eight nodes (see Figure 1(a)).

N1	Receiving from N2 Forwarding to	N7
N7	N5	Null
Null	N7	Null
Null	Null	Null

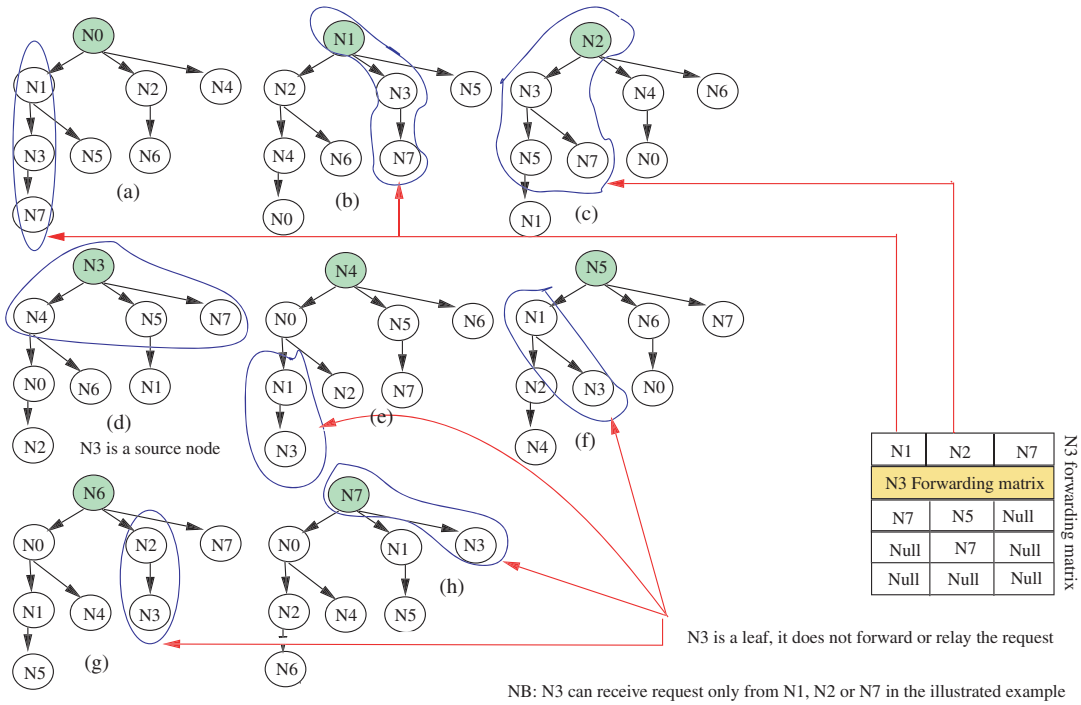

Figure 10. Multi-tree for many-to-many applications in Chord-based GPM with eight nodes.

When a node N_i receives and accepts request *Child* (N_k) from its predecessor, it puts N_k on the first entry of its forwarding matrix, and when it forwards the request *Child* (N_i) for discovering its immediate successors in the associate tree, N_i adds the successors in the same column of N_k.

As shown in Table III, when node N3 receives data from node N1, it forwards to node N7 (*column 1*), when it receives data from N2, it forwards to nodes N5 and N7 (*column 2*), and when it receives data from N7 (*end receiver*), it discards the received data. Algorithm 4 illustrates the forwarding process for delivering data from node N_i to the appropriate nodes.

Example

Figure 10 shows an example of multicast trees computed for multiple sources on Chord-based GPM. These multicast trees are stored in the GPM system as a set of multiple matrices (*forwarding matrix such as* Table III), each matrix being constructed and associated with one node. As shown in the figure, the source multicast trees corresponding to all possible multicast sessions are represented. The forwarding matrix reflects a part of these trees for a particular node, used locally for relaying the flow. For instance, N3 forwarding matrix shows that it can receive requests only from nodes N1, N2 and N7 (*this constitutes the first line of the forwarding matrix*). When a node is a source, it forwards data to all its children in the finger table as shown in Figure 10(d) corresponding to line 2 of Algorithm 4. However, when a node is a leaf of a tree in the corresponding multicast

DOI: 10.1002/dac

Algorithm 4 : Media forwarding algorithm

1: Begin
2: If (N_i *is a source node*) **then**
3: Send data to all children in the finger table.
4: else //N_i *is not a source // it receives data from node* N_t *(e.g. column c on Tab)*
4.1: begin
4.2: For j=0 **to** $(Num-1)$ **do**// *Num is the degree of the underlying network, generally it is equal to* $log_2(n)$ *in the structured P2P network.*
4.3: if $((Tab[j,c] \neq Null)$ and N_i is not a "leaf" in the tree **then**
4.3.1: Forward data to the node Tab [j,c]
4.4:End
5: End.

session (Figure 10(e)–(h)), it stops forwarding the received flow. When node N3 receives data from node N2, N3 participates to the multicast session with N2 as source (*as illustrated in* Figure 10(c)), it forwards to both nodes N5 and N7. When it receives from node N1 (*in case where N3 participates to the multicast session with N0 or N1 as a source or participates to the two sessions simultaneously*), it forwards to node N7 in any case.

Algorithms composition.

Previous and current sections describe several algorithms. These algorithms comprise the overall GPM approach. Figure 11 gives a flow chart that describes the algorithm interactions for both one-to-many and many-to-many type of applications. At the GPM initialization, the adjacency matrix is built and stored at each node. For one-to-many type of applications, only one multicast tree is considered. In this case, a source node initializes a new multicast session through Algorithm 3 (*line 2*). At the reception of a request child(N_j), relay nodes execute line 4 of the same algorithm. The generated multicast tree is used for flow distribution. At a join or leave requests, the multicast tree will be updated (*see* Section 3.2) but also the finger tables using the stabilization algorithm of the underlying overlay network, and consequently the adjacency matrix using Algorithm 2. The multicast session will not be re-initialized (*Algorithm 3 is executed only for a new multicast session*).

For many-to-many type of applications, multiple multicast trees are considered simultaneously. Algorithm 3 initialized by a source node is executed for each multicast session (*like in one-to-many type of applications*), then a forwarding matrix (e.g. Table III) will be constructed for each node, and updated at each join or leave request. Based on this matrix, Algorithm 4 is used to distribute the data flow.

5. GPM PERFORMANCE EVALUATIONS

For the GPM performance evaluation, we consider the following metrics as defined in [25] for evaluating application layer multicast:

- *Data path quality*: Two metrics are defined: (a) *Stress*: which is defined per link and counts the number of identical packets sent by the protocol over that link or node. For ALM, there is no redundant packet (*duplication*); the stress metric is 1 for each network link. (b) *Stretch*: it is defined per member, as the ratio of the path length (*along the overlay from the source to the member*) to the length of the direct unicast path.
- *Tree depth*: It measures the distance from a source node to the far end node. Note that tree depth has a significant impact on the overlay latency.
- *Overhead*: Each member exchanges refresh messages with its neighboring peers on the overlay. These messages constitute the control traffic (*overhead*) generated by the multicast group.
- *Convergence time*: It measures the time needed for multicast tree construction.

DOI: 10.1002/dac

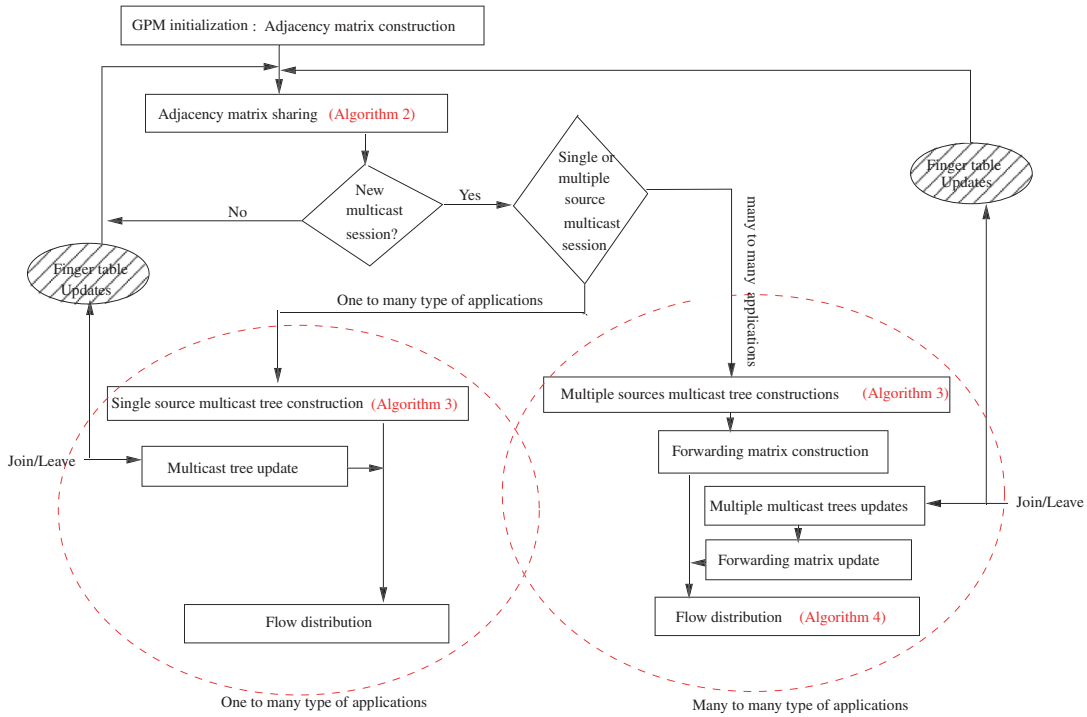

Figure 11. Chart flow of GPM functional principle for one-to-many and many-to-many type of applications.

For evaluation purpose, we consider two scenarios corresponding to: a structured (e.g. *Chord illustrated in* Figure 1(a)) and unstructured (e.g. *Gnutella illustrated in* Figure 1(b)) P2P overlay. We also assume that the cost between each neighboring node is one hop. The simulations are down on machine carried out in a personal computer with the following characteristics: 2.16 GHz and 1GB of RAM. A specific tool[‡] was developed for simulation purposes. Simulation tool (*version 1.0*) provides different performance metrics for GPM, such as Convergence time, tree depth and traffic overhead under certain conditions (e.g. *number of nodes, type of overlay network, churn rate, etc.*). Figure 12 gives an illustration of the user interface and capture screen of the simulation tool, when considering the construction of a multicast tree based on a network with 50 nodes and N3 as a source node.

5.1. Stretch evaluation

We generate a Chord topology and randomly a Gnutella topology based on different scenarios (*with different number of nodes*). We calculate the shortest path from source node N0 to each receiver node, from both generated overlay network topology and the generated multicast trees. Then we calculate the average stretch. Tables IV and V provide two examples.

Table IV illustrates the average stretch in a Chord-based GPM (*see* Figure 1(a)), for the optimized (a) and basic (b) approaches. The results show that the average stretch obtained from the optimized approach (1) is lower compared to the basic approach (1.12). The shortest path from N0 to N5 is 2 hops on Chord, and it is equal to 3 (*resp 2*) hops on the generated multicast tree from basic (*resp optimized*) approach. The same phenomenon is observed on path from N0 to N7. This is essentially due to the GPM algorithm that computes a multicast tree, where the number of hops from source to any receiver node is equal to those of the underlying overlay network.

Table V illustrates the average stretch in a Gnutella-based GPM (*see* Figure 1(b)), for the optimized (a) and basic (b) approaches. The results show that the average stretch obtained from

[‡]www.telecom-lille1.eu/people/meddahi/mourad/P2pGpmSim.htm.

DOI: 10.1002/dac

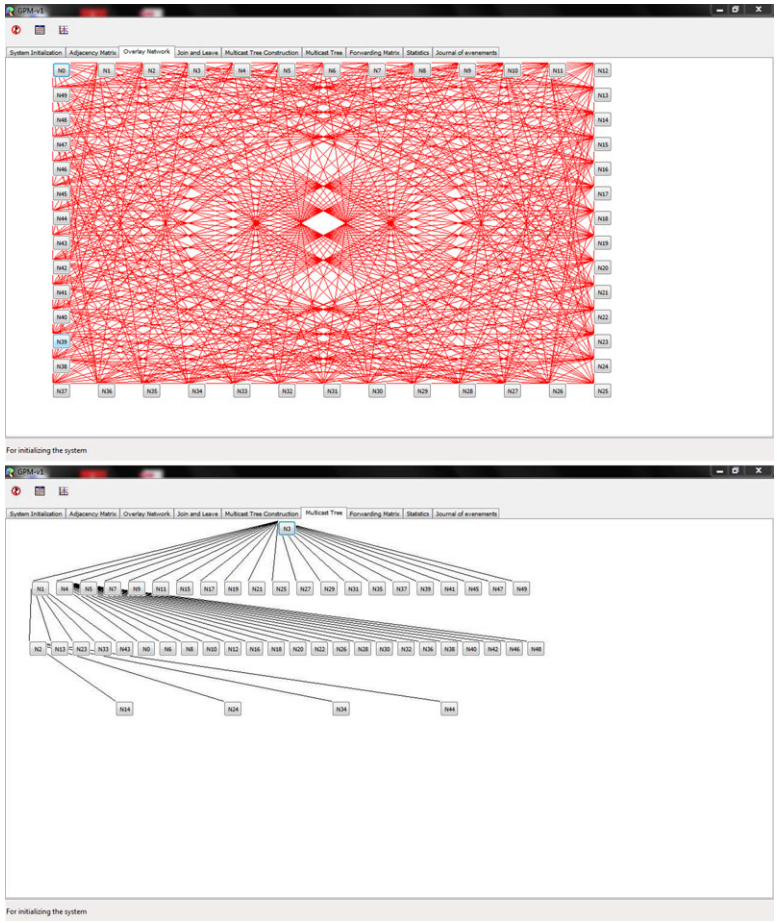

Figure 12. Chord-based GPM network with 50 nodes, and the associated multicast tree with N3 as a source node.

Table IV. AVG Stretch with the optimized (a) and basic approaches (b) for a structured P2P networks (Chord).

Nodes link	Overlay unicast path	GPM multicast path	Stretch
Links	(a) and (b)	(a) and (b)	(a) and (b)
N0 → N1	1	1—1	1—1
N0 → N2	1	1—1	1—1
N0 → N3	2	2—2	1—1
N0 → N4	1	1—2	1—1
N0 → N5	2	2—3	1—1.5
N0 → N6	2	2—2	1—1
N0 → N7	3	3—4	1—1.33
Avg. stretch		1 (a)—1.12 (b)	

the optimized approach (1) is lower compared to the basic one (1.06). The shortest path from N0 to N6 in Gnutella network is 3 hops. However, the same path on the multicast tree generated by the basic (*resp. optimized*) approach is 4 (*resp. 3*) hops. The same phenomenon is observed on path from N0 to N11.

Considering the GPM approach, the stretch for both trees is equal to one. As a conclusion, GPM is independent of the underlying network in terms of average stretch which is always equal to one (*we note that the average stretch equal to one means that there is no better tree*).

DOI: 10.1002/dac

Table V. Avg stretch with the optimized (a) and basic approaches (b) for an unstructured P2P network (Gnutella).

Nodes link	Overlay unicast path	GPM multicast path	Stretch
Links	(a) and (b)	(a) and (b)	(a) and (b)
N0 → N1	1	1—1	1—1
N0 → N2	2	2—2	1—1
N0 → N3	1	1—1	1—1
N0 → N4	2	2—2	1—1
N0 → N5	2	2—2	1—1
N0 → N6	3	3—4	1—1.33
N0 → N7	4	4—5	1—1
N0 → N8	5	5—6	1—1
N0 → N9	3	3—3	1—1
N0 → N10	4	4—4	1—1
N0 → N11	5	5—7	1—1.4
Avg. stretch		1 (a)—1.06 (b)	

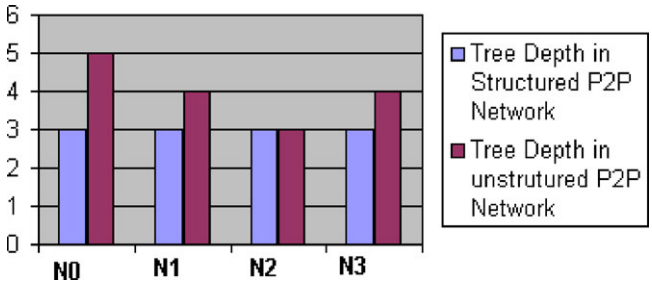

Figure 13. Tree depth in Chord- and Gnutella-based GPM with eight nodes when N_i is a root.

Tables IV and V show that GPM is generic and independent of the underlying network (*structured or unstructured P2P network*), if the stretch is constant and equal to one, then the multicast tree is more efficient.

5.2. Tree depth and end-to-end delay evaluation

Figure 13 shows the tree depth for both Gnutella- and Chord-based GPM obtained from the optimized approach when N_i is a source. In Chord-based GPM, the average stretch is $O(\log_2(n))$. As an example, when eight nodes are considered, the tree depth is 3. However, in Gnutella (*unstructured P2P*) based GPM, the tree depth not only depends on the source node position in the network, but also on the current network overlay. When node N2 is a source, it constructs a multicast tree (*depth=3*) better than (*in terms of depth*) if N0 is a source (*tree depth=5*).

Figure 14 shows the multicast tree depth as a function of the number of nodes in the network. It grows logarithmically, when using optimized approach in the case of structured P2P networks. On average, it is equal to $O(\log_2(n))$, where n is the number of nodes in the network. This scenario shows that the GPM approach is well adapted to the structured P2P overlay, where the tree depth (*overlay end-to-end latency*) is steady and independent of the source node location, and then scalability is reinforced. However, in unstructured P2P network, the GPM multicast tree depth depends on the the source node location in the network.

The generated multicast tree for structured P2P network is balanced compared to unstructured P2P networks, because GPM takes into consideration the characteristics of the underlying architecture. As a conclusion, when GPM is applied to structured P2P network and from a source node perspectives, each source node 'see' the same QoS.

Figure 15 presents the average end-to-end delay as a function of the number of nodes for both GPM and the basic approach when Chord and Gnutella architectures are used. In [34], the authors measure 137 ms as the value between two neighboring openDHT nodes. In our simulation, we

DOI: 10.1002/dac

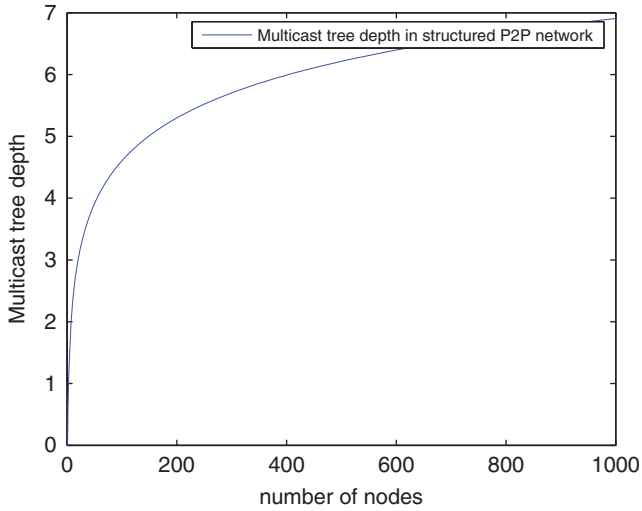

Figure 14. Multicast tree depth as a function of the number of nodes in Chord-based GPM.

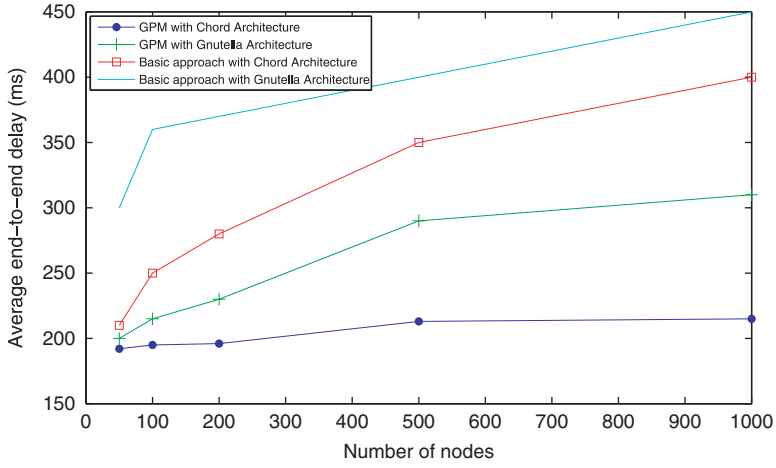

Figure 15. End-to-end delay as a function of the number of nodes.

generate 100 random values in the interval [130 160] and we take the average. This figure shows that the average end-to-end delay for GPM is better controlled than for the basic approach (*such as MCAN*) on any type of P2P networks. The utilization of Chord architecture for both GPM and the basic approach optimizes the end-to-end delay comparatively to Gnutella architecture. As an example, for 200 nodes, the average end-to-end delay for GPM (*resp. basic approach*) when using Chord architecture is 196 ms (*resp.* 280 ms). However, it is equal to 230 ms (*resp.* 370 ms) when using Gnutella architecture. The optimization of the end-to-end delay is a consequence of the tree depth optimization which is re-enforced on the structured P2P architecture such as Chord. We can also notice that the average end-to-end for the GPM approach is below 300 ms which is under the threshold defined by the ITU-T G113-114 recommendation for voice quality.

5.3. Convergence time evaluation evaluation

Figure 16 illustrates the convergence of the multicast tree construction algorithm for Chord (*resp. Gnutella*) based GPM, as a function of the number of nodes in the network. As an example, for 12 nodes, the convergence time of GPM algorithm for constructing a multicast tree when N1 is a source node is 10 ms for Chord-based GPM (*resp.* 8 ms *for Gnutella-based GPM*). For 1000 nodes, the convergence time is 354 ms for Chord-based GPM, and 277 ms for Gnutella-based GPM. The

DOI: 10.1002/dac

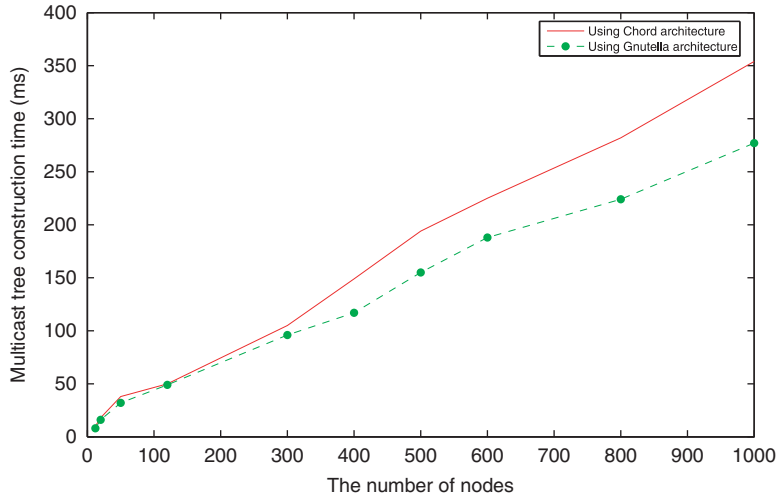

Figure 16. Convergence time for Chord- and Gnutella-based GPM.

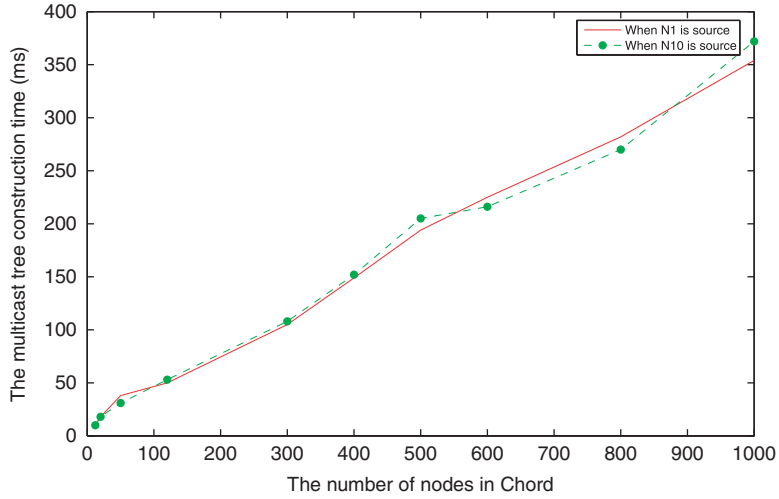

Figure 17. Convergence time for Chord-based GPM when N1 (*resp. N10*) is a source and for different number of nodes.

results show that for both structured and unstructured P2P architecture-based GPM (*optimized approach*), the convergence time increases linearly as a function of the number of nodes. This is because the multicast tree construction is distributed, while enabling the parallel computation of independent parts locally on each node (*greedy decision aspect of GPM algorithms*). Both scalability and operability of the proposed GPM are better improved and enhanced.

Figure 17 (*resp.* Figure 18) characterizes the convergence time (ms) for Chord- (*resp. Gnutella*) based GPM with two scenarios: when N1 is a source node and when N10 is a source node (*the Gnutella network is randomly generated*). Convergence times as illustrated in the figure are almost equivalent when different sources are considered. Consequently, both structured and unstructured P2P network-based GPM (*optimized approach*) are independent of the source node position for GPM convergence. This characterizes the stability of GPM for the underlying overlay network.

5.4. Overhead evaluation

Figure 19 shows the number of messages generated (*control traffic overhead*) for the multicast tree construction process on Chord-based GPM, for both basic and optimized approaches. As an example, for the basic approach and when 12 nodes are considered, the number of generated

DOI: 10.1002/dac

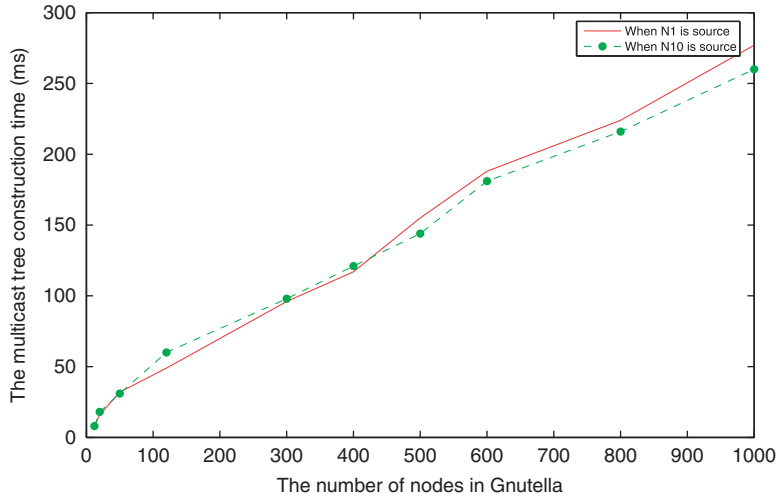

Figure 18. Convergence time for Gnutella-based GPM when N1 (*resp. N10*) is a source and for different number of nodes.

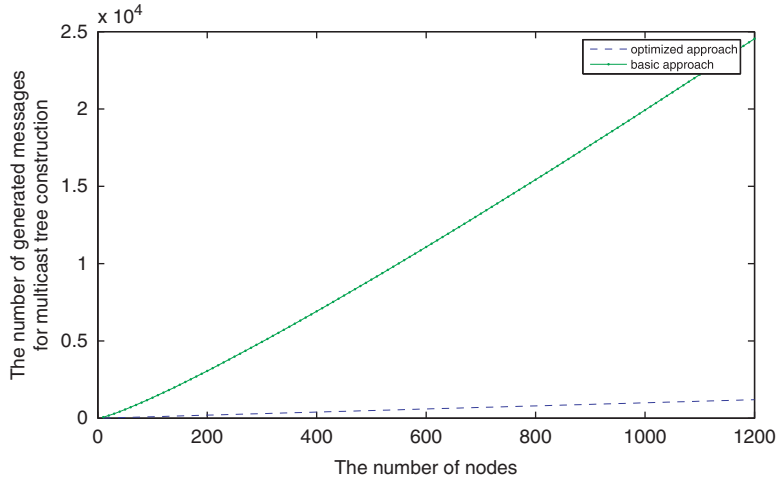

Figure 19. Overhead for multicast tree construction process for Chord-based GPM.

messages is 72, while for the optimized approach, only 11 messages are needed. For 500 nodes, the basic approach generates 8000 messages, and the optimized one generates 499 messages for building a multicast tree. The main reason comes from the characteristics of Algorithm 3 that reduces flooding messages. At each relay node in the multicast tree, due to the greedy algorithm, only the nodes that are not in the multicast tree or will not be part (*with deterministic manner, see line 6 of Algorithm 3*) are considered. This scenario shows that optimized approach (GPM) gives a better control of traffic overhead than the basic approach. Because the multicast tree generated from the optimized approach is more balanced in terms of depth and width, as node sends messages only to a few number of nodes (*generally* $\log_2(n)$ *nodes on the structured P2P network*). As a result, the flooding mechanism is limited.

5.5. Data structure analysis

Figure 20 shows both the adjacency and the forwarding matrix sizes as a function of the number of nodes. Both matrices are cost effective in terms of memory size, as they are implemented as a matrix of bits. As an example, when 256 nodes are considered, the size of the adjacency matrix is

DOI: 10.1002/dac

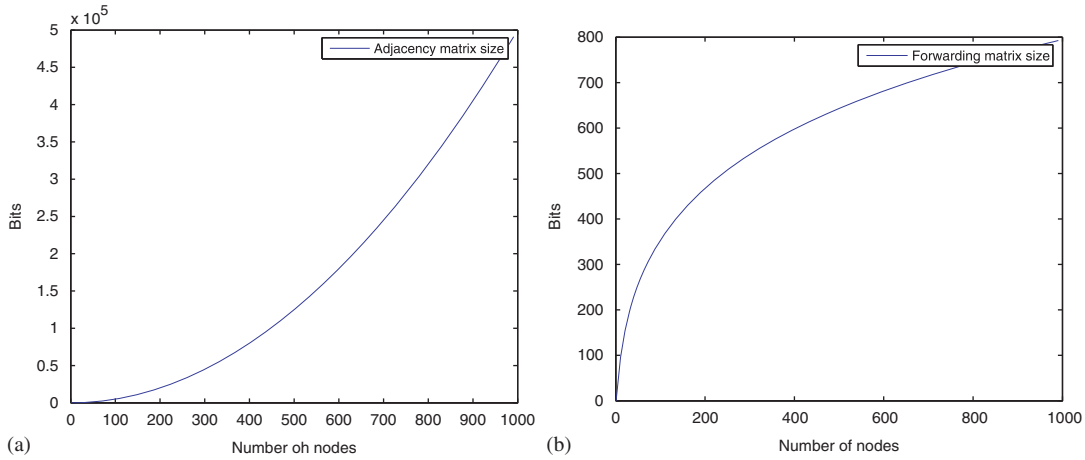

Figure 20. Adjacency vs forwarding matrix sizes.

65 536 bits (*8 kB*), and the size of the forwarding matrix is 72-node identifiers (*576 bits= 0,07 kB*). So their impact on GPM performance is limited, which is a key point, particularly for nodes with limited resources. This scenario illustrates the GPM operability as limited resources are required for its implementation. In case of high capabilities nodes, scalability increases.

6. CONCLUSION AND FUTURE WORKS

On the one hand, heterogeneity is one of the main attributes that characterize the Internet, and on the other hand, there is a tremendous development of P2P-based IP services such as P2P IPTV or more generally group-oriented services as in [35]. For this type of multicast application or services, not only the physical delay or latency constitutes a critical parameter in terms of QoS, but also the latency at the overlay (ALM) should be controlled. Given this, it is necessary to create an efficient, fault-tolerant and scalable solution for flow distribution, while controlling network latency, especially for group-oriented communications. Existing ALM-based applications use a different, independent and specific (*non-generic*) mechanisms for multicast tree construction.

The approach we propose is generic and scalable, while optimizing the multicast tree depth, and consequently the end-to-end overlay delay for both classes of P2P networks (*structured and unstructured*). From a basic intuitive approach that minimizes complexity, but not really cost effective in terms of traffic overhead and overlay latency. The proposed approach is generic, it can be implemented on any existing P2P overlay and benefits from its advantages. GPM is more adapted to the structured P2P network as the multicast tree is balanced, it makes use of particular data structures called adjacency matrix and forwarding matrix that can be implemented on nodes with limited capabilities (*PDA*, *mobile phone*, *etc*.). For better considerations, a queuing mechanism such as proposed in [36] could be used at each node. This generic and cost-effective (*overhead, tree depth*) GPM approach for one-to-many applications is extended to support many-to-many type of applications. Performance evaluations show that GPM is characterized by high scalability (*stretch equal to one, tree depth and delay are much optimized*), but also operability, since the specific data structures are cost effective. Most parts of the multicast tree are calculated in a decentralized and parallel way, while the convergence time is improved.

We envision several perspectives for this work. One perspective is to implement and experiment GPM in the context of a P2P multi-conference system developed in a previous work [37]. To evaluate GPM in a real and critical P2P testbed, we also plan to deploy the GPM approach on the PlanetLab platform. Another perspective is to optimize some GPM functionalities. A first optimization is related to the bootstrapping process: the bootstrapping server could store a copy of the multicast tree topology in order to maintain the tree optimized even when a new node joins the

DOI: 10.1002/dac

network. A second optimization will consist in extending the adjacency matrix in order to reflect the physical characteristics of the underlying network (*network delay*, *loss*, *etc.*) and exploit them in GPM algorithms.

ACKNOWLEDGEMENTS

The authors thank Mrs N. Khebbache and Mrs O. Assoul from Bejaia University for their comments and suggestions.

REFERENCES

1. Yeoa CK, Leea BS, Erb MH. A survey of application level multicast techniques. *Computer Communications Journal* 2004; **27**:1547–1568.
2. Zhang B, Wang W, Jamin S, Massey D, Zhang L. Universal ip multicast delivery. *Journal of Computer Networks* 2006; **50**(6):781–806.
3. Fahmy S, Kwon M. Characterizing overlay multicast networks and their costs. *IEEE/ACM Transactions on Networking* (*TON*) 2007; **15**(2):373–386.
4. Steinmetz R, Wehrle K. *Peer-to-Peer Systems and Applications*, vol. 3485. Springer: Berlin, 2005.
5. Rodrigo R, Peter D. Peer-to-peer systems. *Communications of the ACM* 2010; **53**:72–82.
6. Zhang B, Jamin S, Zhang L. Host multicast: a framework for delivering multicast to end users. *IEEE Infocom*, New York, U.S.A., June 2002.
7. Pendarakis D, Shi S, Verma D, Waldvogel M. Almi: an application level multicast infrastructure. *The Third USENIX Symposium on Internet Technologies*, San Francisco, CA, U.S.A., March 2001.
8. Hsiao H-C, He C-P. A tree-based peer-to-peer network with quality guarantees. *IEEE Transactions on Parallel and Distributed Systems* 2008; **19**(8):1099–1110.
9. Mol JD, Epema DHP, Sips HJ. The orchard algorithm: building multicast trees for p2p video multicasting without free-riding. *IEEE Transactions on Multimedia* 2007; **9**(8):1593–1604.
10. Jiang T, Zhong A. A multicast routing algorithm for p2p networks. *GCC'04*. Lecture Notes in Computer Science, vol. 3032. Springer: Berlin, 2004; 452–455.
11. Castro M, Druschel P, Kermarrec AM. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications* 2002; **20**(8):1489–1499.
12. Ratnasamy S, Handley M, Shenker S. Application-level multicast using content addressable networks. *Proceedings of the International Workshop on Networked Group Communication* (*NGC*), London, U.K., November 2001.
13. Gnutella. Available from: http://www.gnutella.com.
14. Napster. www.napster.com.
15. Xu X. A cluster-based protocol for resource location in peer to peer systems. *Journal of Parallel and Distributed Computing* 2005; **65**:665–678.
16. Yu J, Li M. Cbt: a proximity-aware peer clustering system in large-scale bittorrent-like peer-to-peer networks. *Journal of Computer Communications* 2008; **31**:591–602.
17. Jiang S, Guo L, Zhang X, Wang H. Lightflood: minimizing redundant messages and maximizing the scope of peer-to-peer search. *IEEE Transaction on Parallel and Distributed Systems* 2008; **19**(5):601–614.
18. Leu JS, Tsai CW, Lin WH. Resource searching in an unstructured p2p network based on cloning random walker assisted by dominating set. *Computer Networks* 2011; **55**(3):722–733.
19. Zhong M, Shen K, Seiferas J. The convergence-guaranteed random walk and its applications in peer-to-peer networks. *IEEE Transactions on Computers* 2008; **57**(5):619–633.
20. Amad M, Meddahi A. P4l: a four layers p2p model for optimizing resources discovery and localization. *Proceedings of APNOMS 2006*. Lecture Notes in Computer Science, vol. 4238. Springer: Berlin, 2006; 342–351.
21. Shen H, Xu CZ, Chen G. Cycloid: a constant-degree and lookup-efficient p2p overlay network. *International Journal of Performance Evaluation* 2006; **63**:195–216.
22. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for internet application. *Proceedings of the ACM SIGCOMM 01*, San Diego, CA, 2001.
23. Tran DA, Nguyen T. Hierarchical multidimensional search in peer-to-peer networks. *Journal of Computer Communications* 2008; **31**:346–357.
24. Lao L, Cui JH, Gerla M. A scalable overlay multicast architecture for large-scale applications. *Technical Report*, *TR040008*, UCLA, November 2004.
25. Banerjee S, Bhattacharjee B. A comparative study of multicast protocols: top, bottom, or in the middle? *Technical Report*, *TR040054*, UCLA, January 2005.
26. Chu Y-H, Rao S, Zhang H. A case for end system multicast. *ACM SIGMETRICS*, Santa Clare, CA, U.S.A., June 2002; 1–12.
27. Banerjee S, Bhattacharjee B, Kommareddy C. Scalable application layer multicast. *ACM SIGCOMM*, Pittsburgh, PA, U.S.A., August 2002.
28. Jannotti J, Gifford DK, Johnson KL, Kaashoek F, O'Toole JW. Overcast: reliable multicasting with an overlay network. *Proceedings of the OSDI*, October 2000; 197–212.

DOI: 10.1002/dac

29. Cramer C, Kutzner K, Fuhrmann T. Bootstrapping locality-aware p2p networks. *Proceedings of the IEEE International Conference on Networks* (*ICON*), Kuala Lumpur, Malaysia, 2005. Available from: http://i30www.ira.uka.de/research/documents/p2p/20.
30. Firooz MH, Ronasi K, Pakravan MR, Avanaki AN. A multi-sender multicast algorithm for media streaming on peer-to-peer networks. *Journal of Computer Communications* 2007; **30**:2191–2200.
31. Do TT, Hua KA, Mounir A. Tantaoui, robust video-on-demand streaming in peer-to-peer environments. *Journal of Computer Communications* 2008; **31**.
32. Enokido T, Tanaka Y, Barolli V, Takizawa M. Distributed multi-source streaming models in peer-to-peer overlay networks. *Journal of Simulation Modelling Practice and Theory* 2007; **15**:449–464.
33. Bikfalvi A, Garcýa-Reinoso J, Vidal I, Valera F. A peer-to-peer iptv service architecture for the ip multimedia subsystem. *International Journal of Communication Systems* 2010; **23**:780–801.
34. Rhea S, Chun B-G, Kubiatowicz J, Shenker S. Fixing the embarrassing slowness of opendht on planetlab. *Proceedings of the Second Workshop on Real*, *Large Distributed Systems* (*WORLDS '05*), San Francisco, U.S.A., 2005.
35. Liu D-K, Hwang R-H. P2broadcast: A hierarchical clustering live video streaming system for p2p networks. *International Journal of Communication Systems* 2006; **19**:619–637.
36. Rajendran RK, Rubenstein D. Optimizing the quality of scalable video streams on p2p networks. *Journal of Computer Networks* 2006; **50**:2641–2658.
37. Meddahi A, Vanwormhoudt G, Malkawi A. A p2p framework for decentralized xconferencing and its jxta implementation. *3PGIC-2008*, *the Second International Workshop on P2P*, *Parallel*, *Grid and Internet Computing*, Barcelona, Spain, March 2008.

AUTHORS' BIOGRAPHIES

Mourad Amad received the Engineer degree from the National Institute of Computer Science (*INI-Algeria*) in 2003, and the Magister degree from the University of Bejaia (*Algeria*) in 2005. Currently, he is a PhD student at the University of Bejaia, Member of the laboratory L.A.M.O.S. His research interests include peer-to-peer networks (*architecture, application, security, VoIP*).

Ahmed Meddahi is a member of Institut Telecom/Telecom Lille I, Computer Science and Networks Department. He obtained his Masters degree from the University of Lille (*France*) and PhD from the University of EVRY (*France*) and 'Institut National des Telecommunications'. His main interests are focused on IP signalling performance, 'VoIP' quality, 'context aware' management and P2P networks.

Djamil Aïssani is Professor at Bejaia University, Algeria, head of the Science and Engineering Department (*1999–2000*). He is the Director of the LAMOS Laboratory (*Modelling and Optimisation of Systems—http://www.lamos.org*), Scientific leader of the Doctoral Computer School (*since its opening in 2003*). His research focuses on Markov chains, queueing systems, reliability theory, inventory, risk theory, performance evaluation and their applications.

DOI: 10.1002/dac

Gilles Vanwormhoudt is a member of the Institut Telecom/Telecom Lille I, Computer Science and Networks Department. He received his PhD in Computer Science from the University of Lille 1. His research interests focus on advanced concepts and techniques for the design and programming of software in distributed systems.

DOI: 10.1002/dac