Distributed Multicast Tree Aggregation

Joanna Moulierac, Alexandre Guitton

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Distributed Multicast Tree Aggregation

Joanna Moulierac and Alexandre Guitton

N°5636

Juillet 2005

Systèmes communicants

ISSN 0249-6399 ISRN INRIA/RR--5636--FR+ENG

Distributed Multicast Tree Aggregation

Joanna Moulierac * and Alexandre Guitton †

Systèmes communicants
Projet ARMOR

Rapport de recherche n ° 5636 — Juillet 2005 — 24Contentssection*.2 pages

Abstract: Multicast is not scalable mainly due to the number of forwarding states and control overhead required to maintain trees. Tree aggregation reduces the number of multicast forwarding states and the tree maintenance overhead by allowing several multicast groups to share the same delivery tree. In this paper, we exhibit several drawbacks of the existing protocols: the latency to manage group dynamics is high, the managers are critical points of failures and some group-specific entries are stored unnecessarily. Then, we propose a new distributed protocol that significantly reduces the number of control messages and limits the number of trees within a domain. By simulations, we show that our protocol achieves good performance and outperforms the previous known distributed algorithm.

Key-words: tree aggregation, aggregated multicast, networks

(Résumé : tsvp)

* joanna.moulierac@irisa.fr
† alexandre.guitton@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Agrégation d'arbres Multicast distribuée

Résumé : Le Multicast n'est pas encore bien déployé dans Internet. Les deux raisons principales qui freinent son déploiement sont : le nombre d'états de routage important qui dépend du nombre de groupes et le nombre de messages de contrôle nécessaires pour maintenir les arbres multicast dans un domaine de routage. L'agrégation d'arbres multicast est un protocole qui permet de résoudre ces deux problèmes en permettant à plusieurs groupes multicast d'utiliser le même arbre de routage. Dans ce papier, nous détaillons plusieurs inconvénients concernant les protocoles réalisant l'agrégation d'arbres. En effet, dans ces protocoles, la latence pour gérer la dynamicité des groupes est grande, les gestionnaires d'agrégation sont des points critiques dans le cas de pannes et des entrées spécifiques aux groupes sont stoquées inutilement. Nous proposons un nouveau protocole distribué qui réduit le nombre de messages de contrôle envoyés et qui limite le nombre d'arbres dans un domaine. Par des simulations, nous prouvons que notre protocole a de bien meilleures performances que le tout dernier protocole distribué connu.

Mots-clé : Agrégation d'arbres, multicast, réseaux

1 Introduction

With the development of Internet, several multimedia applications including group communications have been deployed. Multicast allows to forward the packets for groups on an efficient way concerning bandwidth. Indeed, instead of sending several unicast packets to the members of the groups, multicast consists in sending only one packet to the address of the multicast group. The packet is duplicated at the branching nodes of the tree covering the group. Unfortunately, multicast is not well deployed over the Internet. One of the main limits of the deployment of multicast is the scale of the forwarding states and the control overhead required to maintain the trees. In unicast, the forwarding states can be aggregated, and in this way the size of the forwarding table is reduced. However, the same solution cannot be applied to IP multicast as the multicast addresses do not reflect the location of the members and because of the group dynamics. Moreover, concerning the tree maintenance for multicast, control messages traverse the network and utilize a large amount of the network resources.

Tree aggregation is a recent proposition that reduces the number of multicast forwarding states together with the tree maintenance overhead. The main idea of tree aggregation is to force multiple multicast groups to share the same delivery tree within a domain. A label, which corresponds to the delivery tree and is significant only in the domain, is added to every multicast packet at ingress routers and removed at egress routers. In this way, the number of forwarding states in the domain depends only on the number of trees instead of on the number of groups. Additionally, the control overhead is reduced since few trees are built.

Protocols achieving tree aggregation have to match groups to trees efficiently and several solutions have been proposed in the literature. However, all the current propositions suffer from several drawbacks. In all these protocols, a centralized manager is in charge of performing aggregations. The routers of the domain request this manager when they have to forward packets for groups and in this way the manager is a critical point of failure. Indeed, when the manager fails, no more aggregations can be achieved. Moreover, upon group changes, the border router that detects the changes systematically requests this manager, which induces latency for the groups.

In this paper, we propose a new distributed protocol for tree aggregation. In addition to reducing the number of forwarding entries and the tree maintenance overhead, our protocol: (i) is much less subject to failures than the existing protocols as it is fully distributed, (ii) induces a minimal latency to deal with group dynamics and (iii) does not store any group-specific entries.

The rest of the paper is organized as follows. Section 2 presents the main protocols achieving tree aggregation and describes the drawbacks of this framework. In Section 3, our protocol DMTA is presented. In Section 4, we discuss about the advantages and the drawbacks of DMTA. The results of the simulations are detailed in Section 5. Finally, the perspectives of our work are given in Section 6.

2 Related Work

Tree aggregation was first proposed in [10] and since, several protocols based on tree aggreagtion have been proposed. For brevity, we will describe here only AM, which is the first centralized protocol, and BEAM, which is decentralized. Other protocols that achieve tree aggregation are AMBTS [4] that performs sub-tree aggregation, STA [11] that performs fast aggregations, ASSM [8] that is designed for source-specific groups, AQoSM [5] and Q-STA [13] that deal with bandwidth constraints, and TOMA [12] that deals with overlay multicast. These protocols are not described in this paper.

2.1 Context of tree aggregation

Tree aggregation reduces the number of multicast forwarding states and the tree maintenance overhead. To achieve this reduction, several multicast groups share the same delivery tree within a domain. Consequently, less trees are built than with traditional multicast. To aggregate several groups to the same tree, a label corresponding to the tree is assigned to all the multicast packets at the ingress routers of the domain. In the domain, the packets are forwarded according to this label. The label is removed at the egress routers so that the packets can be forwarded outside the domain.

In addition to the multicast forwarding states that allow to match an incoming label to a set of outgoing interfaces, the border routers of the domain have group-specific entries. These states match group addresses to labels and they are stored in a group-label table.

2.2 AM: a centralized protocol

Aggregated Multicast (AM) was proposed in [10, 6]. In AM, a centralized entity called the tree manager is in charge of matching groups to labels and informing the border routers of this matching. This tree manager knows the set of all the multicast trees configured in the domain.

Let us consider the domain depicted on Figure 1. When the border router b_3 receives a `join` message for the group g (message 1), it sends a request to the tree manager (message 2). The tree manager knows that b_1 and b_4 have already joined the group g. This manager searches for a tree t for the aggregation of g (algorithm 3a). The tree t has to cover the routers b_1, b_3 and b_4 and has to satisfy bandwidth constraints. If a candidate tree for g is found, g is matched to the label of the tree. If there is no candidate tree for g, the tree manager configures a new tree in the domain and adds it in the multicast tree set after having assigned a label l to this new tree.

Once a label is matched to a group, the tree manager informs the border routers attached to members of the group of the matching of g to l (message 4). They all add a group-specific entry $g \rightarrow l$ in their group-label table. The process is similar when a `leave` message is received.

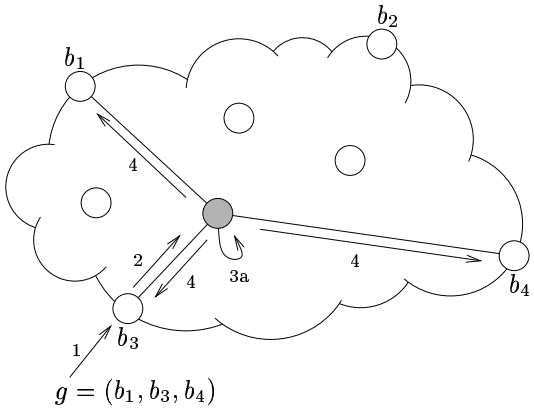


Figure 1: Tree aggregation with AM.

2.3 BEAM: a decentralized protocol

Bidirectional Aggregated Multicast (BEAM) was proposed in [7]. To the best of our knowledge, BEAM is the only decentralized protocol that achieves tree aggregation. In BEAM, the task of the tree manager is distributed among several routers, called core routers. The set of multicast trees is also balanced among these core routers. Let us consider the domain depicted on Figure 2. When the border router b_3 receives a `join` message for the group g (message 1), it sends a request to the dedicated core router in charge of g (message 2). This dedicated core router (obtained using a hash function) searches for a tree for g in its own multicast tree set (algorithm 3a). If this initial core router does not find a candidate tree, it requests all the other core routers (message 3b). They search in their own multicast tree set (algorithm 3c) and reply to the initial core router if they have found a candidate tree (message 3d). Then, the dedicated core router chooses among all the answers the best candidate tree for g (algorithm 3e) or builds a new tree. Finally, the core router informs all the border routers attached to members of g of the matching $g \rightarrow l$ (message 4). The border routers add the group-specific entry $g \rightarrow l$ in their group-label table.

The mechanism used by BEAM is similar to the mechanism used by AM, except that the multicast tree set is balanced among all the core routers. It can be noticed that algorithm 3c and algorithm 3e in BEAM are faster than algorithm 3c in AM, since fewer trees are evaluated.

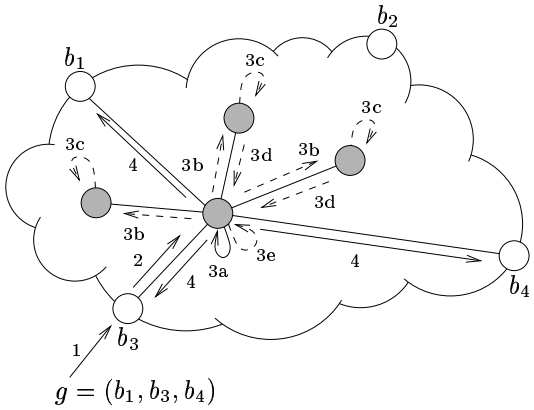


Figure 2: Tree aggregation with BEAM.

2.4 Drawbacks of the existing protocols

The protocols that achieve tree aggregation suffer from three main drawbacks.

First, the tree manager and the core routers are critical point of failures. In AM, if the centralized tree manager fails, packets can still be forwarded but no new group can be accepted in the domain and the group memberships are unable to change (messages `join` and `leave` would not be processed). In BEAM, if a core router fails, all the groups associated to this core router are binded to their current trees.

Second, the management of the groups induces significant control overhead. In AM (respectively BEAM), any membership change induce requests to the tree manager (respectively to the core routers) (message 2), the search for a candidate tree (algorithm 3a) and the reply to all border routers (message 4). All these actions increase the latency for the groups. In BEAM, there is additional control with the requests to all the core routers (message 3b), the search for a candidate in their own multicast tree set (algorithm 3c), their replies (message 3d) and the candidate selection (algorithm 3e).

Third, the existing protocols build group-specific entries in border routers. Indeed, the matching of a group g to a label l is stored in all the border routers attached to members of g. In the context of the reduction of forwarding states, such entries should be avoided because their number scale with the number of groups. Indeed, tree aggregation reduces the number of forwarding states in core routers, but one group-specific entry per group is added in the border routers.

In the next section, we present our new protocol that solves these drawbacks.

3 DMTA: Distributed Multicast Tree Aggregation protocol

Most of the drawbacks of the aggregated multicast framework come from the centralized entity which induces control overhead and is inherently a critical point of failure. This entity is required since storing the multicast tree set in all the border routers is impossible due to memory requirements. Our proposition DMTA (Distributed Multicast Tree Aggregation) does not require any centralized entity as DMTA is fully distributed.

The main idea of our protocol is to enable the border routers to match groups to labels. In this way, they can forward the multicast packets without any request to centralized entities. Then, the control overhead is reduced. In DMTA, labels are significant, thus the border routers know which routers are covered by a label corresponding to a tree. In this way, the border routers are able to search for a label for a group in the list of labels available in the domain.

Consequently, in DMTA, there is need of neither tree manager nor core routers as in AM or BEAM. This section describes firstly the algorithm DMTA and secondly the way the list of labels available in the domain is set up by a topology manager.

3.1 Distributed Multicast Tree Aggregation algorithm

The border routers keep a list of the labels available in the domain in order to match groups to labels. This list of labels is built by a topology manager and this construction is detailed later in section 3.2. In this section, DMTA algorithm is described formally.

3.1.1 Description of the algorithm

DMTA is a new distributed tree aggregation protocol. In this protocol, when a border router b receives a packet for a group g, b finds a label l for g using Algorithm 1. Basically, b searches for an adequate label l for g in its label set $LS(b)$. Note that only a subset of $LS(b)$ is evaluated if the labels are sorted according to the number of spanned nodes and if only the labels which cover at least $|g|$ nodes are evaluated[1]. Then b forwards the multicast packet by encapsulating it with the label l. In DMTA, the border routers do not maintain any group-specific entries and a label is searched for each incoming multicast packet.

Note that the border routers have the knowledge of the group memberships in the domain. This information can be retrieved by several means usually with MOSPF [14]. If MOSPF is not deployed, additional control messages are needed. When a border router b receives a `join` (resp. a `leave`) for a group g, b notifies all the border routers of the domain that it is attached to members of g (resp. that all the members of g attached to b have left). Then, all the border routers update the group membership of g. However, the same knowledge is required in any tree aggregation protocol. AM and BEAM keep also this information in the tree manager and in the core routers.

[1] Additional information may be found in [11].

Input: A multicast packet for the group g reaches the router b
Output: b finds a label for g in its label set $LS(b)$
  $C \leftarrow \emptyset$
  for all $l \in LS(b)$ **do**
    if l can cover g **then**
      add l to C
  choose the label in C which covers the less number of nodes

Algorithm 1: Group label matching algorithm of DMTA.

In DMTA, the border routers do not keep any group-specific entries. In this way, no specific action is required for each `join` or `leave`. Indeed, the border routers will simply find another label for the next incoming packet. In AM or BEAM, the tree manager has to find a new label for the group and all the border routers have to update the corresponding group-specific entries.

3.2 The set of labels available in a domain

This subsection describes how the topology manager builds the set of trees within a domain and how a label corresponding to a tree can be significant to a border router. In DMTA, the border routers know which labels are available in the domain, *i.e.*, which trees are configured. We introduce a new entity, the topology manager, which is in charge of building the set of trees, setting up the corresponding list of labels and informing the borders routers of the set of labels available. The topology manager is only active when configuring the trees during the initialization process and when topology changes occur. Apart from these moments, the topology manager has no interaction with the aggregation process. Indeed, it does not take any decision concerning the aggregation of the groups. Therefore, it cannot be compared to the tree manager as the tree manager is in charge of the aggregation of the groups.

3.2.1 How many trees for a domain?

The set of available trees is computed off-line by the topology manager and is specific to the topology of the domain. In a domain with B border routers, there are 2^B different multicast groups. Protocols achieving tree aggregation assume that in the worst case, 2^B different trees have to be maintained. The number of trees is expected to be huge within a large domain and it is not possible to configure them all due to memory requirements.

However, this number can be greatly reduced when considering that a single tree can cover several groups without wasting bandwidth. As an example, note that the tree represented on Figure 3 is the native tree of the groups with members in (b_1, b_3, b_4). This tree is also the native tree (*i.e.* the one with the minimum number of links) for the group (b_1, b_4). Indeed, these two groups use this tree to deliver multicast packets. Consequently, only one tree is built for these two groups and the number of different trees is lower than the number of different groups. Recall that bidirectional trees are built.

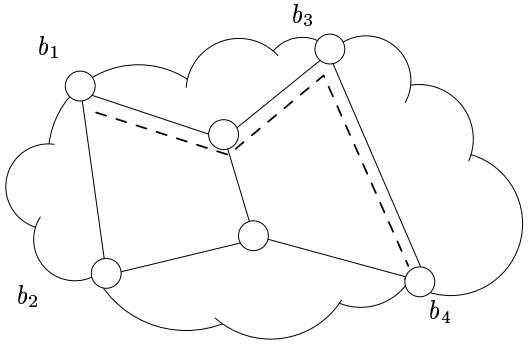


Figure 3: The tree represented on the figure covers two different multicast groups.

Input: A domain with B routers
Output: A set of trees $\mathcal{T}$ covering all the 2^B groups
  $\mathcal{T} \leftarrow \emptyset$
  for $i = 1$ to 2^B **do**
    g_i is the i-th group of the 2^B possible groups
    Compute a native tree t_i covering g_i
    if there is no tree in $\mathcal{T}$ that covers exactly the same routers as t_i **then**
      add t_i to $\mathcal{T}$
  return $\mathcal{T}$

Algorithm 2: Construction of the set of different trees for a domain with B routers

Algorithm 2 computes the set $\mathcal{T}$ of different trees for a domain with B border routers. The 2^B different groups are generated, and a native tree for each group is built. The native trees are added to $\mathcal{T}$ if no tree in $\mathcal{T}$ already covers exactly the same routers. Then, at the end of the algorithm, each tree in $\mathcal{T}$ can be identified by the set of border routers it covers and each tree is unique in $\mathcal{T}$.

We made simulations of Algorithm 2 on different real graphs in order to compute the number of different trees. The results of these simulations follow.

- On the Abilene network [1], which has 11 nodes and 14 edges, there are 2048 different groups and only 131 different trees.
- On the Nsfnet network [3], which has 14 nodes and 21 edges, there are 16384 different groups and only 958 different trees.
- On the Geant network [2], which has 18 nodes and 30 edges, there are 262144 different groups and only 8222 different trees.

Then, we made simulations of Algorithm 2 on realistic random topologies, generated using the Waxman model (with parameters $\alpha = 0.25$ and $\beta = 0.2$), with the number of nodes varying from 10 to 18. The results are shown on Figure 4 using a logarithmic scale. Each point of the figure is the average on 100 topologies. The figure shows that the number of different trees is several order of magnitudes lower than the number of different groups.

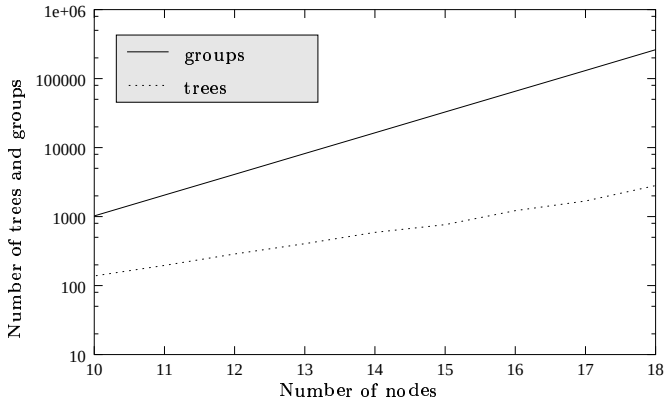


Figure 4: The number of different trees is several order of magnitudes lower than the number of different groups.

The number of different trees is very low compared to the number of different groups. On Table 1, for Nsfnet network, only 958 trees are needed to cover any group. It is realistic to configure these trees off-line. On a larger network, such as the Geant network, 8222 trees are required. If the network is too large and if the number of different trees is too high, the topology manager can remove one of two trees that are similar, *i.e.*, with few different links.

	Number of different groups	Number of different trees	Mean number of labels kept by routers
Abilene network (11 nodes)	2048	131	62
Nsfnet network (14 nodes)	16384	958	522
Geant network (18 nodes)	262144	8222	4534

Table 1: The border routers keep only few labels in their label set.

The topology manager erases the smaller tree in that case. Bandwidth may be wasted but less trees will be configured and maintained.

3.2.2 Assigning a significant label to a tree

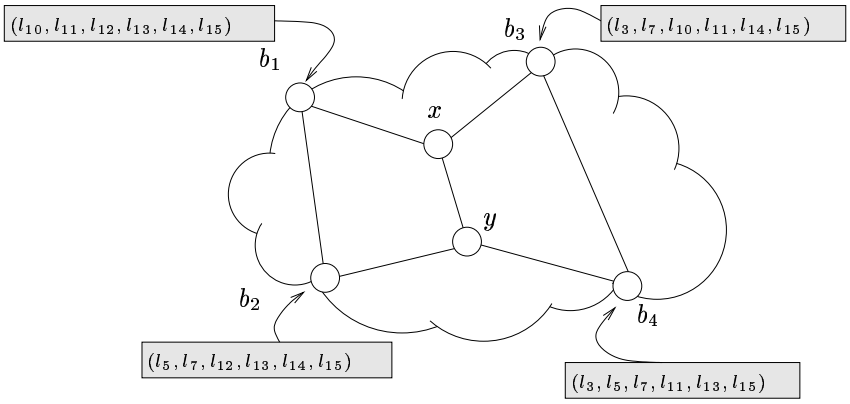
The set of the trees $\mathcal{T}$ is computed by the topology manager. Then, each tree in $\mathcal{T}$ is assigned a unique label corresponding to the covered routers. Indeed, each tree t in $\mathcal{T}$ covers a unique set of routers in a domain with B border routers and can be identified by a label represented by a bitmap of size B. The i-th border router of the domain corresponds to the i-th bit in the bitmap. This i-th bit of the bitmap is set to 1 if the i-th border router of the domain is covered by the tree t and is set to 0 if not.

For example, on Figure 5, there are four border routers: b_1, b_2, b_3 and b_4. The trees within the domain are assigned labels represented by bitmaps of size 4. The label l_5 is assigned to the tree t with links (b_2y, yb_4). The tree t covers b_2 and b_4 and covers neither b_1 nor b_3. Therefore, t is assigned the label number 5 which is the decimal value of the bitmap 0101.

On Figure 5, the border routers store different lists of labels. Indeed, the topology manager send to the border routers only the labels whose trees cover them. For example, on Figure 5, 9 trees are built for the domain but the topology manager send to the border routers only a subset of all the labels configured. In this way, the border routers do not keep the list of all the available labels in the domain. Therefore, the size of the list maintained by the border routers is reduced as seen on the third column of Table 1 and there are less trees to evaluate for the group label matching algorithm.

3.3 DMTA on an example

In this subsection, DMTA is described on the example depicted on Figure 5. A domain with 4 border routers is represented on this figure. Each border router b has received from the topology manager its own list of labels $LS(b)$. For example, b_3 keeps the list $LS(b_3) = (l_3, l_7, l_{10}, l_{11}, l_{14}, l_{15})$.



$$\left(\begin{array}{ccccccc} l_3 = 0011 & , & l_7 = 0111 & , & l_{10} = 1010 & , & l_{11} = 1011 \quad ... \\ \uparrow & & \uparrow & & \uparrow & & \uparrow \\ (b_3 b_4) & & (b_2 y, y b_4, b_4 b_3) & & (b_1 x, x b_3) & & (b_1 x, x b_3, b_3 b_4) \end{array}\right.$$

Figure 5: In AM or BEAM, the tree manager maintains the set of the trees and their topologies whereas in DMTA, only a list of labels is stored.

Suppose that the border router b_3 receives a packet for the group g. The router b_3 knows that g has members in b_3 and in b_4. Therefore, the group g may be represented by the bitmap 0011. The router b_3 searches for a label for g in its label set $LS(b_3)$. A label l is a candidate if the corresponding tree covers the members of g. This can be determined simply by a binary function: $l \vee (\neg g) = 1111$ means that l is a candidate. Therefore, the candidate labels are $C = (l_3, l_7, l_{11}, l_{15})$. Indeed, neither $l_{10} = 1010$ nor $l_{14} = 1110$ can cover the group: $l_{10} \vee (\neg g) = 1010 \vee 1100 = 1110$ and $l_{14} \vee (\neg g) = 1110 \vee 1100 = 1110$. Oppositely, l_3 can cover the group since $l_3 \vee (\neg g) = 0011 \vee 1100 = 1111$. The border router determines in this simple way if a tree can cover a group or not.

Consequently, the label l_3 is chosen for g as it covers the smallest number of nodes: l_3 covers only 2 nodes while l_7 and l_{11} cover 3 nodes, and l_{15} covers 4 nodes. Then b_3 forwards the packet for g in the domain with label l_3. Recall that b_3 does not store any group specific entry for g.

3.4 Summary

In this section, we briefly summarize the main ideas of our protocol. By using significant labels, border routers are able to perform aggregations locally. A topology manager is in charge of configuring and maintaining the trees and informing the border routers of the labels available. Therefore, the task of the topology manager is limited to the initial construction of the trees (the trees are built off-line) and to their reconfiguration when failures occur in the domain.

A border router aggregates a group g to the label that covers the members of g and that covers the minimum number of nodes. In this way, there are less control overhead as there are no requests to centralized entities and less latency for the groups. In DMTA, no group-specific entries are stored.

4 Discussion

In this section we discuss about the group specific entries, the control overhead of aggregation algorithms, the latency for the groups and the robustness of the protocol.

4.1 The group specific entries

In previous propositions, a group specific entry for a group g is stored in the group-label table of the routers attached to members of g. Indeed, the storage of such entries is needed as requests to centralized entities are not possible for each multicast packet of groups.

In our proposition DMTA, there is no requests to centralized entities. Therefore, as our algorithm for selecting a label for a group is fast, there is no need to store group-specific entries. Indeed, a label is searched for each multicast packet and the memory of routers is spared. Moreover, no updates of the group-label tables are needed for each group change as no specific entries are stored. Indeed, in AM or BEAM, for each `join` or `leave` the border routers must update the corresponding group specific entries by using the group label matching algorithm and by flooding the domain with control messages.

4.2 Control overhead

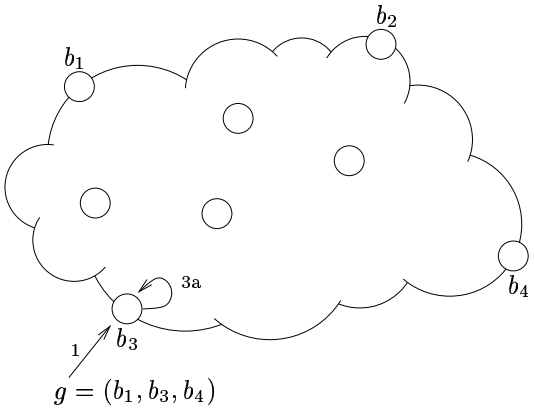


Figure 6: Aggregation with DMTA protocol.

The control overhead denotes the number of control messages sent to and by the tree manager or the core routers. These control messages utilize the network resources and induce latency for groups. Therefore, it is important to minimize the number of control messages sent. Figure 6 shows the aggregation performed with DMTA protocol. Concerning the control overhead, since there is neither tree manager nor core routers, the messages 2, 3b and 3d in Figures 1 and 2 are removed. Furthermore, in our protocol, there is no need for

	AM and BEAM	DMTA
For every packet of a group g	Look-up in the group-label table of size around $\|G\|$	Search for a label in LS with Algorithm 1
For each `join` or `leave`	Request to centralized entity(ies) (messages 2), search for a label with algorithm 3a and update the group specific entries of the border routers (message 4)	Nothing to do

Table 2: Comparison of the behavior of the algorithms AM, BEAM and DMTA

the border routers to synchronize their group-specific entries (message 4 in Figures 1 and 2). The number of control messages is therefore greatly reduced in DMTA.

In AM, when receiving a `join` or a `leave` message, the tree manager searches for a new label for the group. Then, the tree manager sends control messages to the border routers in order to update the corresponding group specific entries. The control overhead is even worse with BEAM because of the additional requests to the other core routers. As DMTA does not keep any group specific entries, no update is needed and no control messages are sent.

4.3 Latency for groups

As DMTA does not keep any group specific entries, there is additional latency for the groups. Indeed, a label is searched for each incoming multicast packet using the group label matching algorithm. This is done in $\mathcal{O}(|LS(b)|)$ for a border router b. In AM or BEAM, the group specific entries are stored and a look-up in this table is done for each packet. Even if the group-label table is large, the look-up can be usually done in $\mathcal{O}(\log |G|)$, where $|G|$ denotes the number of concurrent groups. The look-up is generally faster than the search for a label.

However, the latency induced by DMTA does not depend on the group dynamics. If the groups are highly dynamic, the group specific entries in AM and BEAM will change frequently. This induces an important latency for the groups as it includes: control messages to the centralized entity, aggregation algorithm (with the computation of a native tree for the group and the computation of the bandwidth overhead for each existing tree in the domain) and control messages from the centralized entity to update the group specific entries. See Table 2 to examine the different behaviors of the three algorithms.

If the groups are static, AM and BEAM induce less latency than DMTA for the groups. Oppositely, if the groups are dynamic, DMTA induces less latency for the groups. In addition, as seen in the previous subsections, DMTA reduces the control overhead: there is a trade-off between the latency and the control overhead.

4.4 Robustness

When link failures occur, the topology manager reconfigures the trees that were disconnected and sends the new list of labels to the concerned border routers. In AM or BEAM, in case of link failures, the centralized entities have to update the entire group-label table of all the border routers. During these updates, the packets for groups may be forwarded with wrong labels.

If the topology manager fails, there is no impact for the aggregation of the groups. The only impact will be if the topology manager fails together with some links. Indeed, the topology manager is obviously not able in that case to reconfigure the set of trees. However, packets for groups using connected trees can still be forwarded.

In AM, if the tree manager fails, then all the aggregation process is interrupted. Indeed, there cannot be any new group or any group changing. The failure of the tree manager impacts on both trees and groups, binding groups to static memberships. Although BEAM is decentralized, if a core router fails, then no groups assigned to this core can change.

DMTA is fully distributed as the failure of a router impacts only on the members attached to this router. Indeed, no more members can send or receive packets via this router. However, the other groups can still communicate and be aggregated to trees.

5 Simulation results

We conducted several simulation experiments on different network topologies. The results on these different topologies were similar and we present here the results on the network Nsfnet, which contains 14 nodes and 21 edges. We assume that all the nodes are border routers (which is the case in the real Nsfnet). In order to model realistic multicast groups, we implemented the node weighted model (as in [9]) with 80% of the nodes having a weight of 0.2 and 20% of the nodes having a weight of 0.6. In this model, nodes with a large weight have an high probability of being members of the groups. The number of concurrent groups reach 20000 and the results are plotted for AM, BEAM and DMTA.

In the following subsections, we plot the number of trees, the number of multicast forwarding states, the number of group-specific entries, the number of control messages and the number of evaluated trees.

5.1 The number of trees

Figure 7 shows the number of trees as a function of the number of concurrent groups. With DMTA, the number of trees is constant since they are built off-line by the topology manager. The 958 trees built correspond to all the different trees for Nsfnet. With AM and BEAM, the number of trees is equal and increases with the number of concurrent groups. It reaches more than 2000 for both protocols. Note that AM and BEAM always build the same number of trees, but the trees may be different. Indeed, in BEAM, the initial core router may find a tree for a group in a subset of the multicast tree set and this tree may be different from the one found by AM.

Before 4000 concurrent groups, the number of trees is lower with AM and BEAM than with DTMA. However, after 4000 concurrent groups, there is a reversal of situation. One advantage of our approach is that the set of trees is static and limited, while with AM and BEAM, the number of trees still increases until the upper bound of 2^B.

5.2 The number of forwarding states

Figure 8 shows the total number of multicast forwarding states. Recall that for a tree t a forwarding entry is stored in each router covered by t. So the number of forwarding states is strongly related to the number of trees and that explains why the results for these two metrics are similar.

5.3 The number of group-specific entries

DMTA does not store any group-specific entries. In this section, only the results of AM and BEAM are described. AM and BEAM store the same group-specific entries: one entry in each member of the group. Table 3 indicates the sum of the group-specific entries in all the border routers. For 5000 concurrent groups, 37000 entries are stored in all the border routers while DMTA maintains 0 entries.

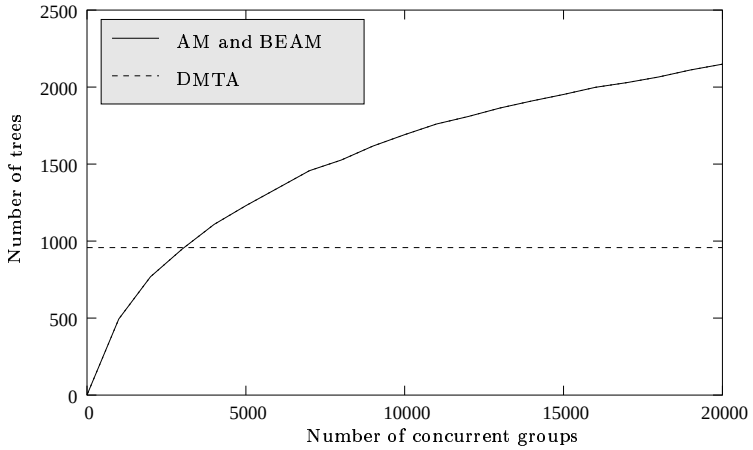


Figure 7: DMTA builds less trees than AM and BEAM.

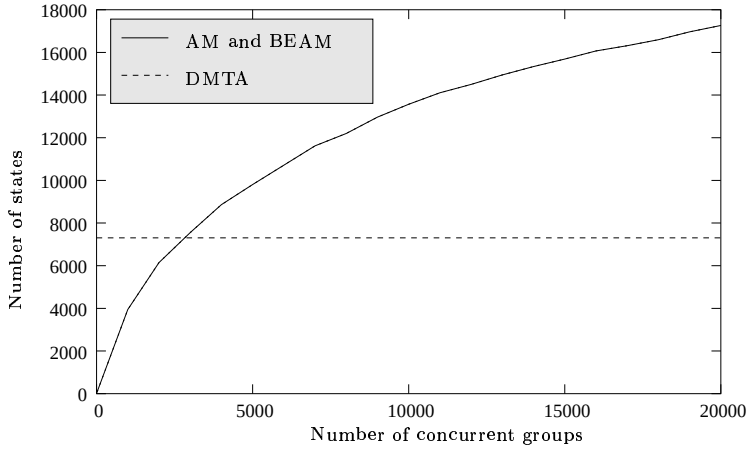


Figure 8: DMTA greatly reduces the number of entries as the number of trees is also reduced.

Number of concurrent groups	5000	10000	15000	20000
Number of group-specific entries for AM and BEAM	37179	74741	112293	149913

Table 3: AM and BEAM store several group-specific entries.

5.4 Number of control messages

The control messages include the requests to the tree manager, and the messages from the tree manager to the members of the group for the update of the group-specific entries. For example, for a new group in AM, there is one control message to the tree manager, and $|g|$ messages in order to configure the group-specific entries where $|g|$ denotes the number of members of the group. As expected, BEAM produces more control messages than AM because of the additional requests to the others core routers. DMTA produces only 14 control messages when the topology manager send messages to the 14 border routers for the initialization of the list of labels available. There are no additional control messages as there are no centralized entities and no group-specific entries.

Consequently, the control overhead in DMTA is significantly reduced and DMTA outperforms AM and BEAM.

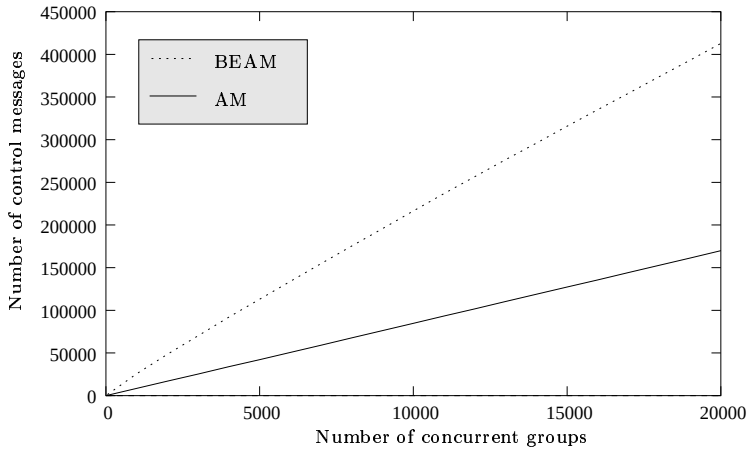


Figure 9: While DMTA generates only 14 control messages (one per border router), AM and BEAM generate several hundred of thousands of control messages.

Figure 10 plots the number of control messages needed to maintain the group memberships. Recall that AM is a centralized protocol and that only the tree manager maintains the group membership. BEAM and DMTA are both distributed and if MOSPF is not deployed in the domain, some additional control messages are needed. Indeed, when a router receives a `join` or a `leave` message, it notifies all the routers of the new (or old) member for the group. DMTA keeps group memberships in all the border routers and BEAM keeps group memberships in all the core routers. As we have assumed that all the routers of the domain are border routers and that in BEAM all the routers may be core routers, they produce both the same number of control messages. AM produces less control messages in that case.

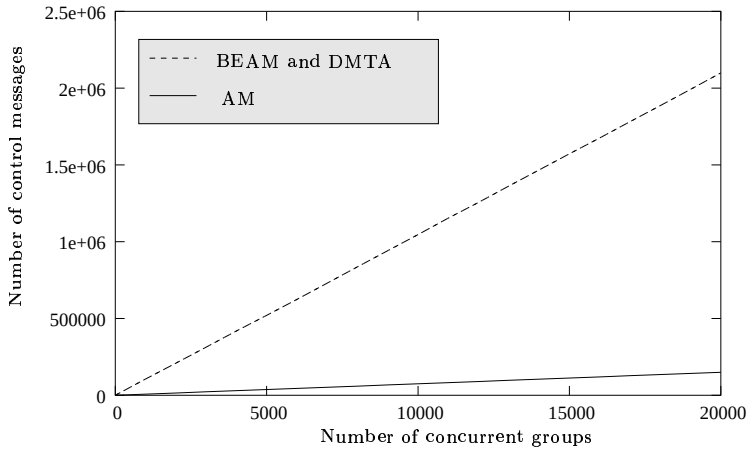


Figure 10: BEAM and DMTA generate more control messages to maintain the group memberships than AM, as they are distributed.

5.5 Number of evaluated trees

Figure 11 shows the total number of evaluated trees for the three algorithms AM, BEAM and DMTA. The number of evaluated trees is

$$\sum_{g \in G} ev(g),$$

where $ev(g)$ denotes the number of evaluated trees for the group g.

As expected, BEAM evaluates less trees than AM as trees are searched in a subset of all the trees in a first time. In DMTA, the border routers evaluate only the trees passing through them. Moreover, the trees are sorted according to the number of spanned nodes and only the trees covering at least $|g|$ nodes are evaluated for a group g. Packets for the group are forwarded once a tree is found. AM evaluates 31 million trees at the end of the simulations while DMTA only 2.2 million trees, that is 14 times less. Consequently, DMTA is faster than AM and BEAM as less trees are evaluated.

5.6 Summary of the simulation results

DMTA is a fully distributed protocol that reduces significantly the number of trees and therefore the number of forwarding states for each router. The set of trees is rather stable as it is built off-line by the topology manager. DMTA induces control messages to maintain group membership in the routers when MOSPF is not deployed. However, the same number of control messages are needed in BEAM which is a decentralized protocol.

DMTA reduces significantly the size of the group-label table and the number of control messages required for the tree aggregation. Indeed, no group-specific entries are stored

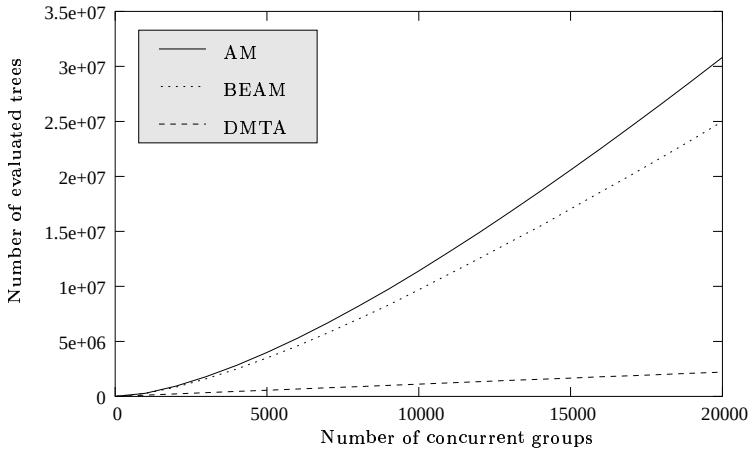


Figure 11: The total number of evaluated trees.

and no request to centralized entities are needed in DMTA. This gain is balanced with an additional latency induced when searching for a label for each received multicast packet. However, when the groups are dynamics, groups in AM and BEAM have high latency when updating the group-specific entries. DMTA behaves in the same way even with an high number of `join` and `leave` messages.

6 Conclusion and future work

Tree aggregation attempts to solve the problem of state scalability and control explosion in multicast by allowing several groups to share the same delivery tree. In this paper, we proposed a new protocol in order to achieve tree aggregation. Our proposition is fully distributed and does not use any centralized entity to perform aggregation, as previous propositions do. Thus, there is no critical point of failures. Additionally, our protocol uses very few control messages, which reduces the latency to deal with group dynamics. Moreover, the number of multicast forwarding states in routers and the number of group-specific entries in border routers are reduced with our proposition.

This work leads to many perspectives of research. Our future work will consist in distributing the topology manager as well, and to investigate deeply topology changes. In case of failures, the topology manager only has to reconfigure the trees that were disconnected and to inform the border routers if the label set is modified. A good solution may consist in building a new set of trees covering exactly the same routers. In this way, it may be unnecessary to reconfigure the label set. We also plan to investigate the construction of the label set in large domains, when the set of all different trees cannot be configured.

Contents

References

[1] Abilene network. http://abilene.internet2.edu.

[2] Geant network. http://www.cybergeography.org/atlas/geant_large.gif.

[3] Nsfnet network. http://www.cybergeography.org/atlas/nsfnet_t1.gif.

[4] F. Cheng and R. Chang. A Tree Switching Protocol for Multicast State Reduction. In *IEEE International Symposium on Computers and Communications*, pages 672–677, July 2000.

[5] J.-H. Cui, J. Kim, A. Fei, M. Faloutsos, and M. Gerla. Scalable QoS Multicast Provisioning in Diff-Serv-Supported MPLS Networks. In *IEEE Globecom*, November 2002.

[6] J.-H. Cui, L. Lao, M. Faloutsos, and M. Gerla. AQoSM: Scalable QoS multicast provisioning in Diff-Serv networks. *Computer Networks*, May 2005.

[7] J.-H. Cui, L. Lao, D. Maggiorini, and M. Gerla. BEAM: A Distributed Aggregated Multicast Protocol Using Bi-directional Trees. In *IEEE International Conference on Communications (ICC)*, May 2003.

[8] J.-H. Cui, D. Maggiorini, J. Kim, K. Boussetta, and M. Gerla. A Protocol to Improve the State Scalability of Source Specific Multicast. In *IEEE Globecom*, November 2002.

[9] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast with Inter-Group Tree Sharing. In *3rd International Workshop on Networked Group Communications (NGC)*, November 2001.

[10] M. Gerla, A. Fei, J.-H. Cui, and M. Faloutsos. Aggregated Multicast for Scalable QoS Multicast Provisioning. In *Tyrrhenian International Workshop on Digital Communications*, September 2001.

[11] A. Guitton and J. Moulierac. Scalable Tree Aggregation for Multicast. In *8th International Conference on Telecommunications (ConTEL)*, June 2005.

[12] L. Lao, J.-H. Cui, and M. Gerla. TOMA: A Viable Solution for Large-Scale Multicast Service Support. In *IFIP Networking*, number 3462 in LNCS, May 2005.

[13] J. Moulierac and A. Guitton. QoS Scalable Tree Aggregation. In *IFIP Networking*, number 3462 in LNCS, May 2005.

[14] J. Moy. Multicast Extensions to OSPF. RFC 1584, IETF, March 1994.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399