Scalable Tree Aggregation for Multicast

Alexandre Guitton, Joanna Moulierac

Scalable Tree Aggregation for Multicast

Alexandre GUITTON and Joanna MOULIERAC
IRISA/University of Rennes I
Campus de Beaulieu, 35 042 Rennes Cedex, France
Phone number: +33 2.99.84.71.30
Email: {alexandre.guitton,joanna.moulierac}@irisa.fr

***Abstract*— IP multicast is not widely deployed yet over Internet. This is mainly due to the forwarding entries scalability and control explosion problems. In this paper, we propose an algorithm called STA (Scalable Tree Aggregation) which reduces the number of trees by allowing several groups to be aggregated to the same tree: the less trees, the less forwarding entries and the less control messages to maintain trees. STA performs faster aggregations than previous aggregation algorithms by evaluating fewer trees for each group, while keeping the same performance. We show the scalability and the fastness of STA by extensive simulations and we compare its performance to the previous algorithm.**

Keywords: scalable multicast, tree aggregation, IP networks.

I. INTRODUCTION

During the last decade, several multimedia applications in Internet involving group communications have been developed. To manage these communications, multicast consists in sending only one message per group instead of unicasting the message to all the members of the group. In traditional IP multicast, each group using services such as audio-video conferencing, Internet video-games, Internet TV or online teaching, is assigned its own multicast address corresponding to a tree delivery structure. As the multicast address of each tree is stored in the forwarding table of the routers spanned by this tree, the number of forwarding entries in the routers increases with the number of trees and consequently with the number of groups.

With the evolution of Internet, the number of groups is expected to grow tremendously, leading to the explosion of the number of forwarding entries. Large forwarding tables slow down IP lookup and then the routing. Indeed, every message for a group involves an IP lookup in the forwarding tables. Moreover, large forwarding tables imply large memory requirements. Additionally, there is an important overhead due to the control messages exchanged by the routers to maintain the trees. Before IP multicast can be widely deployed over Internet, one must solve this problem of scalability.

Several solutions have been proposed in the literature to reduce the size of the forwarding table: the compression of the routing table [1], the use of reduced trees [2], [3], the aggregation of the forwarding entries [4], [5] and tree aggregation [6], [7], [8].

In Reunite [2] and HBH [3], multicast forwarding entries are stored only in branching routers instead of in all the routers of the tree: the number of forwarding entries is reduced. This approach gives good performance when groups are sparse but it does not scale with the number of groups.

In [4], Radoslavov *et al.* propose the aggregation of forwarding entries. Two forwarding entries are aggregated to the longest covering prefix if they share the same incoming interfaces, the same outgoing interfaces and if there is no other forwarding entry that matches the longest covering prefix. They also propose a leaky aggregation for low-bandwidth groups in which bandwidth may be wasted. Leaky aggregation results into a bandwidth-memory trade-off. In [5], Thaler and Handley analyze the aggregation of the forwarding entries formally.

However, these mechanisms do not reduce control overhead due to maintenance of trees, that is why we focus on tree aggregation.

Tree aggregation reduces the number of trees by forcing multiple groups to share the same tree within a domain: these groups are said to be aggregated to this tree. By this way, only one forwarding entry per router is needed for all these groups instead of one per group and per router. Thus, the number of forwarding entries, depending on the number of trees, is reduced. Moreover, less trees are maintained. Different groups may be aggregated to the same tree and since all the leaves of the tree receive the messages, some bandwidth is wasted when leaves with no members receive messages unnecessarily.

Fig. 1 depicts a backbone with four routers: A, B, C and D. A tree t with three links (AB, BC and BD) is represented with dotted lines. Two groups g_1 (with members in A, B, C and D) and g_2 (with members in A, C and D) can use t for their communications. If a group g_3 (with members in A, B and D) use t, bandwidth is wasted when messages for g_3 reach C unnecessarily. Indeed, the link BC is in excess for g_3. Note that the tree spanning g_3 has two links: AB and BD.

Without tree aggregation, three entries are used in router B to maintain the three trees t_1, t_2 and t_3 of g_1, g_2 and g_3. With tree aggregation, if g_1 and g_2 are aggregated to t, only two entries in B are needed for t and t_3. If g_1, g_2 and g_3 share t, only one entry in B is needed for t but bandwidth is wasted for g_3, as explained previously.

The algorithms proposed for tree aggregation [7], [8] were not simulated for large scale multicast with tens of thousands concurrent groups. Indeed, they do not scale with the number of trees and do not address important points of large scale computing such as memory requirements and computation time. In this paper, we

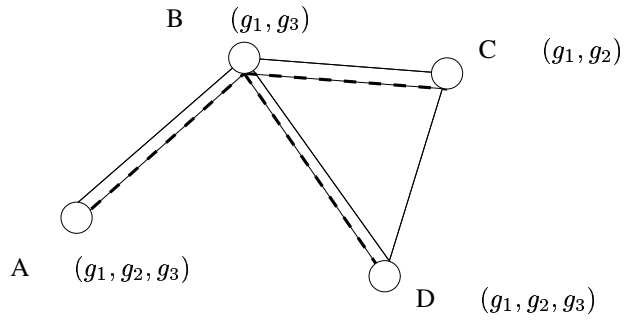

Fig. 1. Within a domain, three groups g_1, g_2 and g_3 can share the same tree t (in dotted lines) but bandwidth is wasted for g_3.

propose a tree aggregation algorithm that aggregates faster than the previous algorithms by evaluating few trees for each group while keeping the same performance as previous algorithms. We define metrics that evaluates the performance of the aggregations and we compare our algorithm to the previous algorithm with around $45,000$ concurrent groups within a large domain.

The rest of the paper is structured as follows. Section II describes the previous algorithms for tree aggregation. Section III exposes our tree aggregation algorithm and the metrics used to define a good aggregation. Section IV presents and analyzes the extensive simulation results.

II. Tree aggregation

Tree aggregation reduces the number of forwarding entries by building less trees than traditional IP multicast. Moreover, decreasing the number of trees reduces the control overhead due to the maintenance of trees.

The tree aggregation idea is proposed by Gerla *et al.* in [6]. Instead of having one tree per group, several groups are aggregated to the same tree within a domain. Messages for these aggregated groups use all the links of the tree: some bandwidth is wasted when messages reach routers without members. Tree aggregation consists in building less trees while controlling the wastage of bandwidth.

In order to have several groups sharing the same tree within a domain, tree aggregation uses labels. At the ingress router, a label is added to each incoming multicast packet and the original multicast address is stored in the packet. The label identifies the tree used by the group receiving the packet. At the egress routers, the label is removed and the original multicast address is restored. In the whole domain, the packets are forwarded according to their label, and not to their original destination address. Using labels is transparent to other domains. Note that the edge routers of the domain maintain the mapping of groups to labels.

To reduce the number of forwarding entries, the tree aggregation algorithms often use bi-directional shared trees.

We describe the following algorithms: Aggregated Multicast (AM) [7] and Bi-directional Aggregated Multicast (BEAM) [8].

A. The centralized protocol AM

A multicast tree set T includes all the trees in the domain. AM [7] decides whether or not to aggregate a new group g to an existing tree in T in the following way: the whole set T is listed and a set C of candidate trees for an aggregation with g is established. A candidate tree satisfies some constraints such as the bandwidth wasted is below a given threshold. AM aggregates g with the tree t in C minimizing a given function $f(g,t)$. If there is no candidate tree, a new tree t_g for g is built and added to T.

In AM, a tree t is a candidate for a group g if $f(g,t) = (C(t)/C(t_g)) - 1$ is less than or equal to the bandwidth threshold t_b where t_g is the native tree of the group g and $C(t)$ is the cost of the tree t.

Each time a new group arrives, AM examines the whole set of trees T. All the trees are systematically evaluated even if they are not likely to be candidates. In large domains with a large number of groups, the number of trees is expected to be important. In this case the approach of AM does not scale with the number of trees.

B. The distributed protocol BEAM

BEAM [8] is a distributed version of AM. In BEAM, each router r keeps its own multicast tree set T^r (r is called a core router). When a group g appears, a core router $r(g)$ is found for g using a hash function, and this router $r(g)$ establishes a set C of candidate trees (satisfying bandwidth constraints) by evaluating the trees in $T^{r(g)}$. BEAM aggregates g with the tree t in C minimizing a function $f'(g,t)$. If no candidate tree is found by $r(g)$ in $T^{r(g)}$, $r(g)$ requests the other core routers to aggregate g with a tree of their own multicast tree set. These requests imply bandwidth wasted because of the generated control messages. As described in [8], the function $f'(g,t)$ for BEAM is faster to compute than the function $f(g,t)$ for AM. Additionally, less trees are listed when a candidate is found in $T^{r(g)}$.

However, BEAM has two main drawbacks. First, all the candidate trees for a new group g are disseminated among all the core routers. Therefore, the probability for a core router to find a candidate in its own multicast tree set is low. Due to the requests of the routers when no candidate tree is found, BEAM often examines the multicast tree sets of all the routers when a new group arrives. This is equivalent to evaluating all the trees in T, as does AM.

The second drawback is that aggregating g with a tree found in $T^{r(g)}$ by $r(g)$ is not necessarily the best decision in terms of bandwidth used considering the whole set of trees T. The aggregation may be suboptimal: two identical groups can have different core routers and can be aggregated to different trees. The number of trees is therefore higher than in AM.

III. Scalable Tree Aggregation algorithm

In this section, we describe our algorithm proposed to achieve fast tree aggregations. Then, we define metrics to evaluate the performance of tree aggregations. We consider the same hypothesis as stated in [7]: links are

never saturated by the multicast groups using that link (the bandwidth allocated for multicast is sufficient) and each group has the same bandwidth requirement. However, we studied tree aggregation with link saturation and with bandwidth requirements for groups in [9].

A. Description of STA algorithm

Recall that T is the multicast tree set. The main idea of STA is to partition the trees in T considering their cost (*i.e.*, the sum of the valuations of the edges of the trees): $T = \{T_1, T_2, ...\}$ where a tree of cost c is in the subset $T_c \subset T$. When a new group g arrives, a tree t_g of cost c_g is computed for g. STA evaluates the trees from the subset T_{c_g} to the subset $T_{c_g + c_g * t_b}$ where t_b is a given bandwidth threshold. As soon as a tree t in these sets can cover g, it is chosen for an aggregation with g. Thus, the aggregation of g is made with the tree in T of minimum cost that can cover g as the trees are evaluated in ascending order of their cost.

The bandwidth threshold t_b for STA determines the maximum bandwidth allowed to be wasted for each group. If $t_b = 0$, no bandwidth is wasted (*i.e.*, the bandwidth used for all the groups remains the same as in traditional IP multicast while the number of trees is reduced). For $t_b = 0.2$, the cost of the candidate tree for a group g is less than 20% of the cost of the original tree t_g of g.

1) Fast aggregations: STA reduces the number of evaluated trees for an aggregation of g: the previous algorithms evaluate all the trees in set T every time a new group arrives, even those of cost far from t_g. In the worst case (no candidate trees for g) STA evaluates only the trees from the sets T_{c_g} to $T_{c_g + c_g * t_b}$. In STA, once a candidate is found, it is chosen for the aggregation whereas the previous algorithms choose a tree among a set C of candidates as shown on Table I (see later).

Algorithm 1 describes STA. Note that how STA determines that a tree t can cover a group g is described later and that the trees have integer costs.

2) Group leaving procedure: When a group leaves, STA checks that the tree for this group is still used, *i.e.*, some groups are still aggregated to this tree. If it is not the case, the tree is removed from the multicast tree set. Otherwise, the tree is maintained.

3) Group changing procedure: When a group changes (*i.e.*, when a member leaves or arrives), STA checks if its aggregated tree still covers the group and if the bandwidth threshold is not exceeded. If one of these check fails, STA considers that the old group leaves (see Group leaving procedure) and that a new group (the old group with the changes) arrives. Then, Algorithm 1 is activated.

Algorithm 1 Scalable Tree Aggregation Algorithm

```
compute a tree t_g for g
t_g has a cost c_g
found ← false and i ← c_g
while not (found) and i ≤ c_g + c_g * t_b do
  while not (found) and there is a tree t in T_i not
  evaluated yet do
    if t can cover g then
      found ← true
    else
      t is evaluated
    end if
  end while
  i ← i + 1
end while
if found then
  aggregate g to t
else {no candidate tree is found considering t_b}
  add t_g in the subset T_{c_g} of T
end if
```

B. Performance metrics

Several metrics are related to the performance of the tree aggregation algorithms.

1) Number of trees: In traditional IP multicast, the number of trees $|T|$ is equal to the number of concurrent groups (each group is assigned its own tree) whereas tree aggregation reduces the number of trees. IP multicast routing protocols maintain each tree by periodically sending control messages and then the less trees in a domain, the less control overhead.

2) Mean number of forwarding entries in a router: Each forwarding entry in a router corresponds to a tree spanning this router. Therefore, the number of forwarding entries depends on the number of trees. A large number of forwarding entries in a router slows down IP lookup, which decreases performance. The mean number of forwarding entries per router is defined as:

$$\frac{\sum_{t \in T} |t|}{N},$$

where T denotes the set of trees, $|t|$ the number of routers spanned by t, and N the number of routers of the domain.

3) Bandwidth wasted: The bandwidth used for a group is considered as the cost of the tree for this group (*i.e.*, the sum of the valuation of each link of the tree). The bandwidth wasted is defined as the ratio of the bandwidth used by our algorithm over the bandwidth used if no aggregation is considered:

$$\frac{\sum_{g \in G} C(t^{STA}(g))}{\sum_{g \in G} C(t_g)} - 1,$$

where G is the set of all the groups, $C(t)$ is the cost of a tree t, $t^{STA}(g)$ is the tree to which g is aggregated by STA and t_g is the original tree for g.

4) Number of trees evaluated per group: To aggregate g to a tree in T, STA evaluates a subset of trees (see Table I). The less evaluated trees, the faster the tree aggregation algorithm. The mean number of evaluations per group is defined as:

$$\frac{\sum_{g \in G} eval(g)}{|G|},$$

where G is the set of all the groups and $eval(g)$ is the number of trees evaluated for the group g.

5) Computation time per group: To prove the scalability of a tree aggregation algorithm, the computation time is important. It corresponds to the time taken by the algorithm to aggregate a given number of groups, considering that there is no group initially and that the group $n+1$ arrives immediately after the group n has been aggregated. The computation time per group is the overall computation time divided by the number of groups. It depends on the number of trees evaluated per group and on the complexity of checking each evaluated tree.

6) Memory requirements: In AM or STA, a *tree manager* is in charge of managing the groups and the trees. To manage several thousands of groups, the tree manager has to store efficiently groups and trees in order to minimize the memory used. Indeed, the memory used is related to scalability, as shown in [10]. The memory requirements also impact on the computation time, due to swapping.

In STA, groups can be stored using bitmaps to reduce the memory requirements. Only the b edge routers of the domain can be attached to members, therefore, a bitmap of size b can identify a group. With $|G|$ groups in the domain, the groups can be stored using $|G|b$ bits of memory. By storing groups with bitmaps, STA can determine quickly if t can cover g using boolean operations. In Fig. 1, the group g_2 is represented by 1011 (members in A, C and D). The tree t (represented by 1111) can cover g_2 because $t \vee (\neg g_2) = 1111$ but a tree t' (represented by 1101 and covering A, B and D) can not cover g_2 because $t' \vee (\neg g_2) \neq 1111$.

IV. Simulations

AM [7] and STA are simulated on the Eurorings network [11], which is a rather large European backbone. It contains 43 nodes and 55 links and all the routers of the domain can be members of multicast groups. We ran these two algorithms on different topologies and the behavior of the algorithms were quite similar. Due to lack of space, we only present here the results for Eurorings network.

Multicast group requests arrive as Poisson process with arrival rate λ and groups lifetimes have an exponential distribution with average μ^{-1}. The average number of concurrent groups is therefore $\frac{\lambda}{\mu}$. We set the arrival rate and the average lifetime in order to get around 45,000 concurrent groups. There were about $350,000$ group requests in our simulations. Note that the STA program can be found at [12].

A. Number of trees

The number of trees measures the efficiency of tree aggregation. The number of trees increases with the number of concurrent groups (see Fig. 2). Additionally, it decreases when the bandwidth threshold increases. The results are approximately the same for both the algorithm: the aggregation ratio of AM is maintained in STA. For example, there are approximately 25,900 trees for 45,000 of concurrent groups for a bandwidth threshold of 0.0 for AM and STA. When considering a scheme without tree aggregation, the number of trees would have been 45,000: tree aggregation achieves in this case around 40% of aggregation ratio.

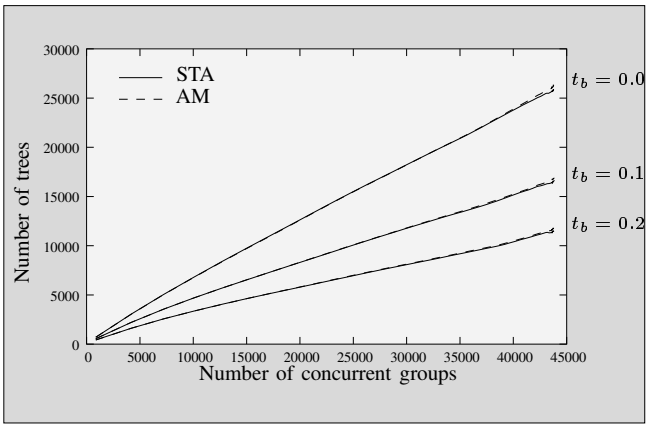

Fig. 2. Number of trees.

B. Mean number of forwarding entries per router

The mean number of forwarding entries per router measures the scalability of tree aggregation. This number increases with the number of groups (see Fig. 3). Additionally, it decreases when the bandwidth threshold increases. For example, there are approximately 17,000 forwarding entries per router for 45,000 concurrent groups when the bandwidth threshold is equal to 0.0 for AM and STA. The mean number of forwarding entries is proportional to the number of spanned nodes by trees and consequently to the number of trees.

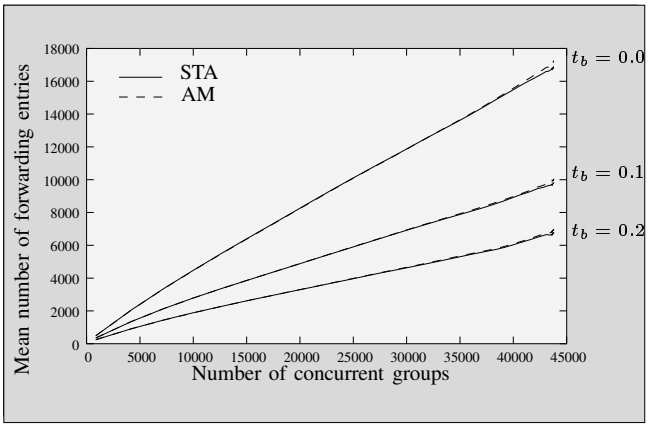

Fig. 3. Mean number of forwarding entries per router.

C. Bandwidth wasted in percent

The more the bandwidth threshold, the more bandwidth is wasted and the less number of trees (see Fig. 4). When the bandwidth threshold is equal to 0.2, around 7% of bandwidth is wasted to maintain approximately 25,900 trees. This percentage is less than the 20% theoretically expected. It appears that the threshold value of 0.2 achieves a good trade-off between the bandwidth wasted and the number of trees.

AM and STA have the same performance for the number of trees, the mean number of forwarding entries per router and the percent of bandwidth wasted. However, STA outperforms AM in the following metrics, which allows to prove the fastness of STA.

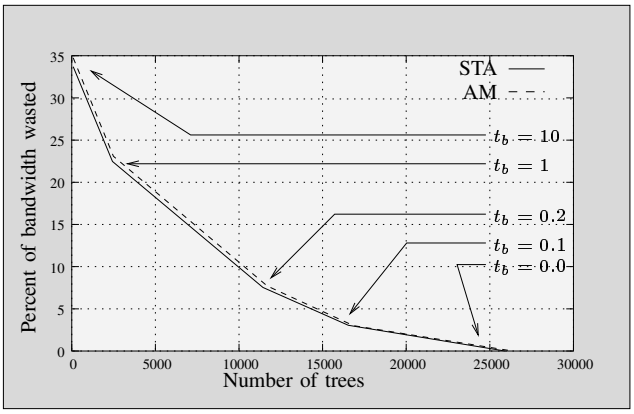

Fig. 4. Bandwidth wasted in percent for 45,000 concurrent groups.

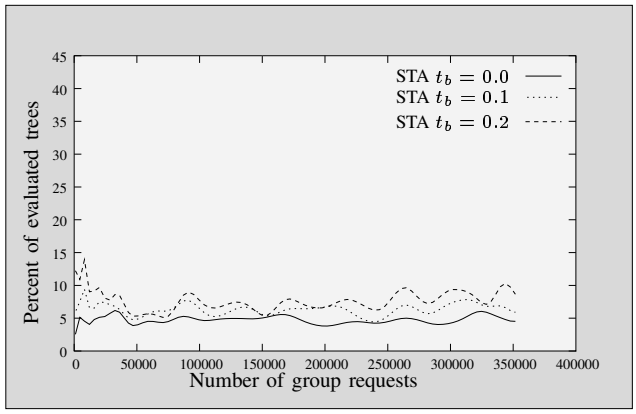

Fig. 6. Percent of evaluated trees per group.

D. Evaluated trees per group request

The number of evaluated trees per group request measures the fastness of a tree aggregation algorithm. The number of evaluated trees per group request increases with the number of group requests (see Fig. 5). Indeed, the more group requests, the more trees and therefore the more evaluated trees. STA evaluates few trees of the multicast tree set for each new aggregation whereas AM evaluates always all the trees of the multicast tree set. For example, for a bandwidth threshold equal to 0.0, STA evaluates 1,000 trees in average for each new group, while AM evaluates in the same time 23,000 trees.

This allow STA to achieve faster aggregations than AM while keeping the same performance.

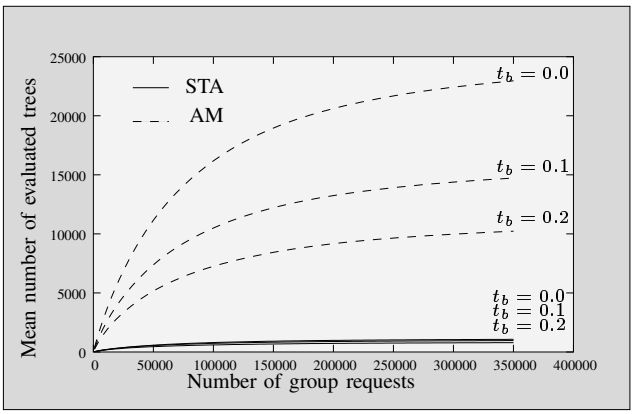

Fig. 5. Mean number of evaluated trees per group request.

Figure 6 shows the percentage of evaluated trees per group request. AM always evaluates all the trees of the multicast tree set, which leads to a value of 100%. STA evaluates always between 5% and 10% of the multicast tree set, whatever the value of the threshold is. The reduction of the number of evaluated trees is significant.

E. Computation time

Our last metric, the computation time, increases with the number of groups (see Fig. 7). Additionally, it decreases when the bandwidth threshold increases. Indeed, when the bandwidth threshold increases, there are less evaluated trees as seen previously.

An Intel Pentium 2.4GHz computer with 1GB memory is used to run the simulations. For a threshold of 0.0, STA takes around 45 minutes to deal with aggregate $350,000$ group requests whereas AM takes more than 3 hours and a half: the computation time is four times shorter in STA. Note that STA is able to aggregate one million of static groups in a short time [13].

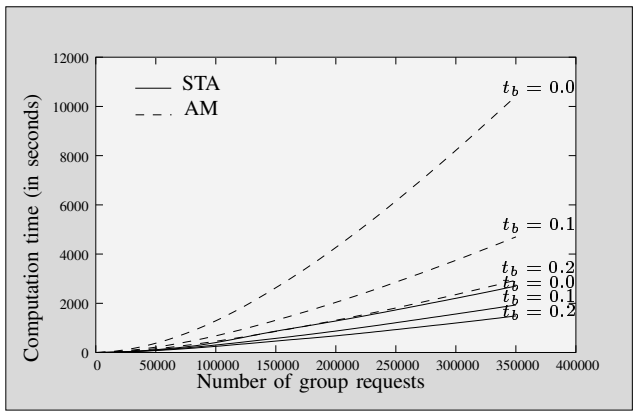

Fig. 7. Computation time (in seconds).

F. Summary of the simulations

In conclusion, tree aggregation is a very promising scheme, which allows to reduce the number of forwarding entries in routers, the number of trees and the control overhead due to the maintenance of trees.

- STA achieves the same performance of aggregation than AM, while reducing consequently the number of evaluated trees for each new group and the computation time (see Table I).
- STA evaluates only trees which are likely to be good candidates and the trees which are not evaluated by STA are not candidate trees for the algorithm AM.

With a bandwidth threshold of 0.0, the number of trees is around $25,900$ and only 7% of bandwidth is wasted for AM and STA.

STA evaluates only $1,000$ trees in average while AM evaluates $23,000$ trees in average for each group request. This reduction of the number of evaluated trees allows to

Metric	AM	STA
Number of evaluated trees per group request g (formally)	$\|T\|$	1 to $\sum_{i=c_g}^{c_g+c_g*t_b} \|T_i\|$
Mean number of evaluated trees per group request ($t_b = 0$)	23,000	1,000
Max number of evaluated trees per group request ($t_b = 0$)	26,369	2,860
Time to manage the $350,000$-th group request ($t_b = 0$)	45 ms	10 ms
Percent of evaluated trees per group request	always 100%	between 5% and 10%

TABLE I

MAIN DIFFERENCES BETWEEN AM AND STA.

speed up the tree aggregations. By this way, STA divides by four the computation time.

V. CONCLUSION

In this paper, we proposed a new algorithm STA to make tree aggregation scalable. The main idea of tree aggregation is to force multiple groups to share the same tree within a domain. STA quickly determines whether a tree is a candidate for an aggregation or not. Moreover, STA evaluates few trees to aggregate a new group. We proposed several metrics to quantify the performance of tree aggregation and we proved the scalability of STA by a comparison with the previous algorithm AM.

Simulation results show that the fast determination of candidates achieves the same performance than the greedy approach of AM for the number of trees, the number of forwarding entries and the bandwidth wasted. Then, our classification of trees and our selection function are relevant. In conclusion STA is easy to implement, runs four times faster than AM while controlling the bandwidth wasted and keeping the same performance.

REFERENCES

[1] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," in *ACM Sigcomm*, 1997.

[2] I. Stoica, T. S. Eugene Ng, and H. Zhang, "REUNITE: A Recursive Unicast Approach to Multicast," in *IEEE Infocom*, March 2000.

[3] L. H. Costa, S. Fdida, and O. C. Duarte, "Hop by hop multicast routing protocol," in *ACM Sigcomm*, August 2001.

[4] P. Radoslavov, D. Estrin, and R. Govindan, "Exploiting the Bandwidth-Memory Tradeoff in Multicast State Aggregation," USC, Dept. of Computer Science, Technical Report 99-697, February 1999.

[5] D. Thaler and M. Handley, "On the Aggregatability of Multicast Forwarding State," in *IEEE Infocom*, 2000, pp. 1654–1663.

[6] 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.

[7] J.-H. Cui, J. Kim, D. Maggiorini, K. Boussetta, and M. Gerla, "Aggregated Multicast — A Comparative Study," *Special issue of Cluster Computing: The Journal of Networks, Software and Applications*, 2003.

[8] 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.

[9] J. Moulierac and A. Guitton, "QoS Scalable Tree Aggregation," in *IFIP Networking*, ser. LNCS, no. 3462, May 2005, pp. 1405–1408.

[10] D. Xu, G. F. Riley, M. H. Ammar, and R. Fujimoto, "Enabling Large-Scale Multicast Simulation by Reducing Memory Requirements," in *Workshop on Parallel and Distributed Simulation (PADS)*, June 2003.

[11] Eurorings, http://www.cybergeography.org/atlas/kpnqwest_large.jpg.

[12] "STA program," *http://www.irisa.fr/armor/lesmembres/Guitton/recherche/sta/sta-en.html*.

[13] A. Guitton and J. Moulierac, "Scalable Tree Aggregation with a Large Number of Multicast Groups," Irisa, Research Report 1663, December 2004.