Hierarchical Aggregation of Multicast Trees in Large Domains

Joanna Moulierac (1) Alexandre Guitton (2) Miklós Molnár (3)
(1) IRISA/University of Rennes I, Rennes, France
(2) Birkbeck College, University of London, England
(3) IRISA/INSA, Rennes, France
Email: joanna.moulierac@irisa.fr, alexandre@dcs.bbk.ac.uk, miklos.molnar@irisa.fr

***Abstract*— Multicast tree aggregation is a technique that reduces the control overhead and the number of states induced by multicast. The main idea of this protocol is to route several groups to the same distribution tree in order to reduce the total number of multicast forwarding states. In this article, we show that this technique cannot be applied to large domains. Indeed, when the number of border routers is large, actual tree aggregation protocols are unable to find similar groups to aggregate to the same tree. However, by dividing the domain into several smaller sub-domains, we prove that it is possible to achieve important savings. A hierarchical protocol is designed to interconnect the trees of the sub-domains together. While previous protocols cannot cope with more than 25 border routers, our protocol still shows significant benefits for domains with 200 border routers.**

I. INTRODUCTION

The growth of multi-users applications such as video-conferences, file sharing, chat rooms or multi-player games is constantly increasing the demand on network bandwidth. For several years, multicast has been considered a solution to save bandwidth by copying packets within the network, rather than at the source. However, multicast has not been deployed on the Internet yet, for it suffers from management issues. First, multicast is not able to aggregate forwarding states efficiently, as unicast does. Therefore, the number of multicast states grows with the number of groups. Second, the control overhead required to manage these states is increasing in the same manner, consuming an important part of the bandwidth. It is not clear that the use of unscalable multicast techniques can really achieve bandwidth savings.

Recently, tree aggregation has been proposed to reduce both the number of multicast forwarding states and its control overhead. While most multicast protocols build a multicast tree per group, tree aggregation uses significantly less trees than groups. Since the states and control overhead depends mainly on the number of trees, the tree aggregation technique can achieve major savings.

This paper is based on "Multicast Tree Aggregation in Large Domains," Joanna Moulierac, Alexandre Guitton, and Miklós Molnár, which appeared in the Proceedings of the 5th IFIP Networking Conference, pp. 691–702, Coimbra, Portugal, May 2006.

A. Tree aggregation technique

Tree aggregation forces several groups to share the same delivery tree by applying a many-to-one function. This function merges all the groups that are similar under the same structure. This matching does not need to be perfect: it is possible that a group g is forced to use a tree t that is sub-optimal for g. The bandwidth loss induced by the messages for the group g is usually not high, because it has to be balanced among all the groups sharing t.

Routing the messages on an aggregated tree cannot be done based on the group address anymore. This problem can be solved by assigning a label to each tree. This label is local to the domain. At the entrance on the domain, this label is pushed into the multicast packets, using a group-label table at the incoming border router. Inside the domain, the packet is routed according to the label only. Finally, at the outgoing border router, the label is removed and the packet forwarding can continue normally outside the domain. In fact, tree aggregation can be deployed in a MPLS domain where the labels are distributed in the domain using LDP or it can be deployed with IP encapsulation where the labels represent a multicast address of a group which is not really active in the domain.

Let us show this mechanism on the domain depicted on Figure 1. The four border routers b_1, b_2, b_3 and b_4 are shown together with the group-label table of router b_1 (this is the only table presented for need of clarity on the figure). Without aggregation, the three groups, with members attached to border routers (b_1, b_2, b_4) are assigned each to one tree. Then, three trees are build and maintained in the domain. When tree aggregation is used, only one tree t_1 can be build for these three groups as the three of them have members attached to the same routers. The tree of label l_1 is configured and three entries that match the three groups to l_1 are added in the group-label table of b_1.

If a new group g_4 with members in b_1 and b_4 joins the network, two possibilities are offered:

- A new tree t_2 can be built for g_4, at the cost of more forwarding states and more control overhead to maintain this new tree. However, if the tree aggregation manager expects more groups similar to g_4 to come, this tree might be proved useful.

- The group g_4 can be aggregated to the tree t_1. In this case, each time a packet for g_4 is sent in the domain, the packet will reach b_2 unnecessarily. In this case, some bandwidth is wasted. Several tree aggregation protocols use the expected bandwidth usage of the group to determine whether to aggregate a group to a tree or not.

Several protocols based on tree aggregation have been proposed in the literature. They yield high benefits. In this article however, we show that tree aggregation cannot achieve significant savings as the size of the domain increases. Indeed, the potential number of groups increases exponentially with the domain size, so when the domain is large enough (*e.g.*, if the domain has more than 25 border routers), the probability that a given number of groups have some commonalities is really small. In the following, we provide an analysis of this behavior.

B. Analysis of the expected number of groups

To show that tree aggregation savings decrease as the size of the domain increases, we conduct a worst-case analysis.

In a domain with b border routers, there are approximately 2^b different groups or more precisely 2^b compositions of groups. Indeed, we consider as a group the set of routers attached with members of groups and with this definition two groups can have two different IP multicast addresses while their members are attached to the same routers. In this case, we say that the two groups are equivalent. Even if the actual number of concurrent groups in the network is higher than 2^b (as there are much than 2^b different multicast addresses), the number of different groups is bounded by 2^b.

Our goal is to derive a formula for the number of expected different groups. Let us identify each concurrent group in the network to a ball, and each possible composition of group to an urn. Each ball is thrown uniformly into one of the urns. The number of different groups $|\mathcal{G}|$ is the resulting number of non-empty urns. If 2^b denotes the total number of urns and g the number of concurrent groups, we have:

$$|\mathcal{G}| = 2^b(1 - (1 - 2^{-b})^g).$$

The previous formula assumes that each possible group has the same probability to appear. This is generally not the case because (i) the probability of a group to appear depends on its size and (ii) multicast groups are often correlated.

Let us apply the formula to compute the number of different composition of groups in a small domain. If the domain contains $b = 15$ border routers, and if there are $g = 10,000$ concurrent groups, the expected number of different groups is only $8\,618$. In such small domains, tree aggregation protocols are applicable where each different group can be assigned a tree. The savings are higher than $1 - |\mathcal{G}|/g = 15\%$ compared to protocols that build one tree per group (in this case there would be $10\,000$ trees). However, if the domain contains $b = 20$ border routers, there are $9\,952$ different groups. Already, the savings of tree aggregation are less than one percent. Notice that the number of different groups is equal to the number of concurrent groups when the number of border routers exceeds $b = 25$ and for a realistic number of concurrent groups. Therefore, it is not possible to have significant savings by applying a tree aggregation technique on a large domain. Table I shows the savings that can be expected by tree aggregation technique as the number of border routers increases.

In this article, our goal is to achieve significant savings in large domains, such as inter-networks constituted of several autonomous systems (AS). We tackle the scalability problem by splitting the domain into several sub-domains and by performing tree aggregation in each sub-domain independently. Then, a centralized manager is in charge of merging the sub-trees together.

C. Splitting the domain increases the aggregation ratio

Let us conduct the same analysis on the number of different groups when the domain is splitted into several sub-domains. Formally, splitting a domain D of b border routers consists in partitioning D into d sub-domains, such as each sub-domain contains approximately b/d border routers. Each sub-domain can be seen as a domain with b/d border routers[1]. Therefore, the expected number of different groups in each sub-domain $|\mathcal{G}_i|$ is equal to:

$$|\mathcal{G}_i| = 2^{b/d}(1 - (1 - 2^{-b/d})^g).$$

The total number of different groups in all the sub-domains is $d.|\mathcal{G}_i|$, which is much smaller than $|\mathcal{G}|$ (roughly, we have $d.|\mathcal{G}_i| \approx d\sqrt[d]{|\mathcal{G}|} << |\mathcal{G}|$).

D. Outline

Section II presents an algorithm that splits the domain into sub-domains, and a protocol that manages the groups dynamically and combines the aggregated trees together. Section III validates the algorithm on simulations, and identifies its advantages according to several metrics. Section V describes the existing protocols for the tree aggregation in small domains. Finally, Section VI provides the perspective of our work and our future directions.

II. The protocol TALD

In this section, we show how to design the protocol TALD (Tree Aggregation in Large Domains) that achieves sub-domains tree aggregation [1]. Three main issues arise in order to present TALD:

- *A.* How to divide the domains into sub-domains?
- *B.* How to aggregate groups within a sub-domain?
- *C.* How to route packets in the whole domain for the multicast group, considering the aggregation of the sub-domains?

The following sub-sections answer to these three questions.

[1]Note that we only count as border routers of the sub-domain the original border routers. The nodes that were not border routers but that are on the boundaries of the sub-domain are not taken into account, since they cannot be attached directly to members.

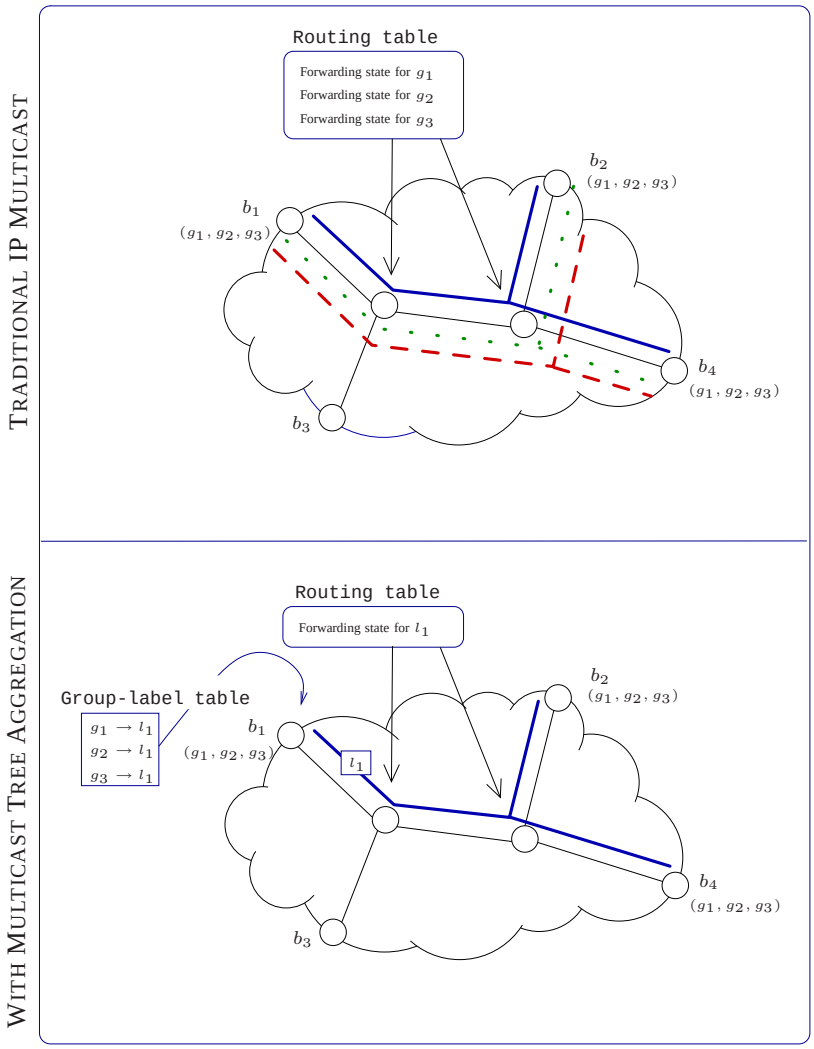


Figure 1. The groups g_1, g_2 and g_3 utilize the same tree of label l_1.

Number of border routers	Number of concurrent groups	Expected number of different groups	Expected savings
15	4 000	3 766	6%
20	4 000	3 992	0%
25	4 000	4 000	0%
30	4 000	4 000	0%
15	10 000	8 618	14%
20	10 000	9 952	1%
25	10 000	9 999	0%
30	10 000	10 000	0%

TABLE I.
THE SAVINGS OF THE TREE AGGREGATION DECAYS RAPIDLY AS THE NUMBER OF BORDER ROUTERS INCREASES.

A. Dividing a domain into two sub-domains

In order to minimize the total number of different multicast groups, the domain D has to be divided into sub-domains D_i of approximately the same number of nodes. We propose Algorithm 1 that divides the domain $D = (V, E)$ into two sub-domains $D_1 = (V_1, E_1)$ and $D_2 = (V_2, E_2)$ where $V_i \subset V$ is the set of routers of the domain D_i and $E_i \subset E$ the set of links.

The main idea of the algorithm is to find first the two nodes x_1 and x_2 with the maximum distance in the domain D, *i.e.* the two most distant nodes. Then, two sets of nodes V_1 and V_2 are created with $x_1 \in V_1$ and $x_2 \in V_2$. Iteratively, the nearest nodes of the nodes already in the set are added. At each step of the algorithm one node is added in V_1 and one node is added is V_2. When all the nodes of the domain D are whether in V_1 or in V_2, two domains $D_1 = (V_1, E_1)$ and $D_2 = (V_2, E_2)$ are built from the two sets. The edges in E_i are the edges including in E connecting two nodes in V_i. When the two sub-domains have been built, this algorithm can be re-applied on each of the sub-domain in order to get 4 sub-domains or more.

Note that in the worst-case, Algorithm 1 can return D_1

Algorithm 1 Dividing a domain into two sub-domains.

Require: a domain $D = (V, E)$
Ensure: two sub-domains $D_1 = (V_1, E_1)$ and $D_2 = (V_2, E_2)$
$(x_1, x_2) \leftarrow$the two most distant nodes in D
$V_1 \leftarrow \{x_1\}$, $V_2 \leftarrow \{x_2\}$
while $V_1 \cup V_2 \neq V$ **do**
  $x_1 \leftarrow$ a direct neighbor of a node of V_1
  $x_2 \leftarrow$ a direct neighbor of a node of V_2
  if x_1 exists **then**
    $V_1 \leftarrow V_1 \cup \{x_1\}$
  end if
  if x_2 exists **then**
    $V_2 \leftarrow V_2 \cup \{x_2\}$
  end if
end while
$D_1 \leftarrow$the sub-graph of D induced by nodes in V_1
$D_2 \leftarrow$the sub-graph of D induced by nodes in V_2

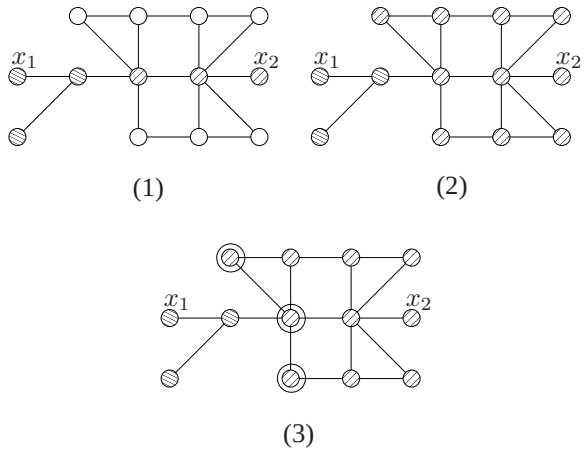


Figure 2. Algorithm 1 is adding three nodes in V_1 and in V_2. However, the nodes of V_2 surround the nodes of V_1, so that V_1 has to be extended with the three encircled nodes.

and D_2 such as $|V_1| << |V_2|$. Such an example is shown on Figure 2. This happens when one of the set cannot be extended because of the other set. In this case, and if the difference of size of the two sub-domains is too high, then the sub-domains can be changed in order to get approximately the same number of nodes in the two sub-domains. This consists in adding in the smallest sub-domain some nodes of the largest sub-domain. On the figure (3) of Figure 2, three nodes of the largest set are added in the smallest set in order to get two equivalent sets. These three nodes are encircled in the figure. When one node is added, the neighbors of this node that are only accessible by it have to be added. When the number of nodes is approximately the same in the two sub-domains (a threshold can be set in order to evaluate this), then the two sub-domains are created.

The same algorithm can be applied separately in each of the sub-domain in order to divide in more sub-domains. Figure 3 shows the network Eurorings [2] divided into four separated sub-domains by the algorithm presented in this subsection. The network was divided into two sub-domains and then they were also divided into two in order to obtain four separated sub-domains with disjoint sets of nodes of approximately the same size.

In this article, we restrict ourselves to $k \in \{1, 2, 4\}$ sub-domains as a proof of concept.

B. Aggregating in a sub-domain

We assume in this subsection that the domain is divided into sub-domains. If the domain is already explicitly divided into small sub-domains (*e.g.*, for administrative reasons), our algorithm can still be applied in these sub-domains if they are too large.

Each sub-domain $D_i = (V_i, E_i)$ is controlled by a centralized entity C_i which is in charge of aggregating the groups within the sub-domain. For clarity of presentation, the aggregation is presented in a centralized fashion but it can also be made in a distributed way. For example, on Figure 3, the sub-domain 1 is controlled by C_1. Each C_i knows the topology of the sub-domain (in order to build trees for the multicast groups) and maintains the group memberships for its sub-domain. Note that C_i is aware of only the members in its sub-domains and not the members for all the group.

When a border router receives a `join` or `leave` message for a group g, it forwards it to the centralized entity C_i in its sub-domain. Then, C_i creates or updates the group specific entries for g in order to route the messages. The centralized entity C_i builds a native tree t_i covering the routers attached to members of g in its sub-domain, and then C_i tries to find an existing tree t_i^{agg} already configured in its sub-domain satisfying these two conditions:

- t_i^{agg} covers all the routers of the sub-domain attached to members of g
- the cost of t_i^{agg} (*i.e.* the sum of the cost of each link of t_i^{agg}) is not more than $b_t\%$ of the cost of the native tree t_i where b_t is a given bandwidth threshold:

$$cost(t_i^{agg}) \leq cost(t_i) \times (1 + b_t).$$

The process of aggregation is described on Figure 4. The border router b_3 which detects a new member for the group g_1 by IGMP messages (step 1), sends a request of aggregation to C_i (step 2). The centralized entity C_i runs the tree aggregation protocol (step 3): C_i chooses among all the trees matching the two conditions described above the tree t^{agg} with minimum cost. If no tree satisfies these two conditions, then C_i configures t_{g_1} (the tree initially built for g_1) by adding forwarding states in all the routers covered by t_{g_1}. Then, C_i sends to all the border routers attached to members of g_1, the group specific entry matching g_1 to t^{agg} or t_{g_1} if no aggregation has been performed. More precisely, this entry matches g_1 to the label corresponding to the tree: $g \rightarrow \text{label}(t^{agg})$ or $g \rightarrow \text{label}(t_{g_1})$.

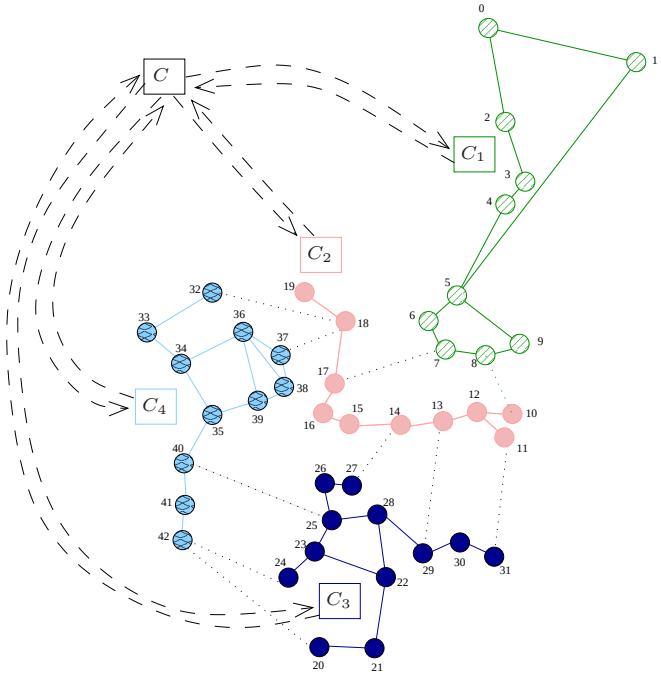
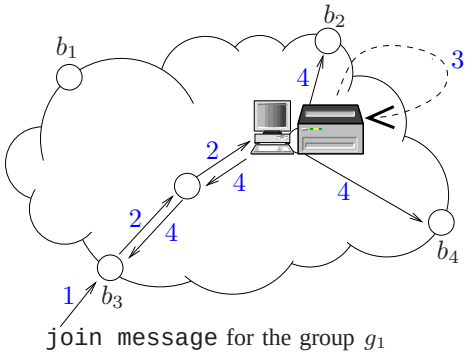


Figure 3. Eurorings network divided into four sub-domains

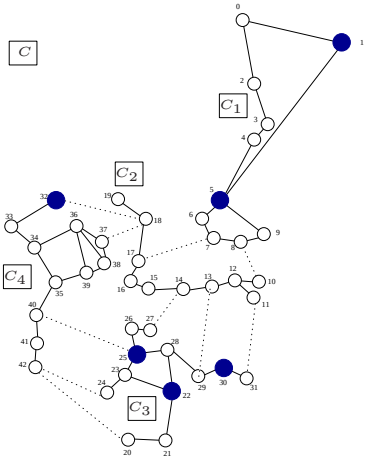


Tables maintained by C_i.

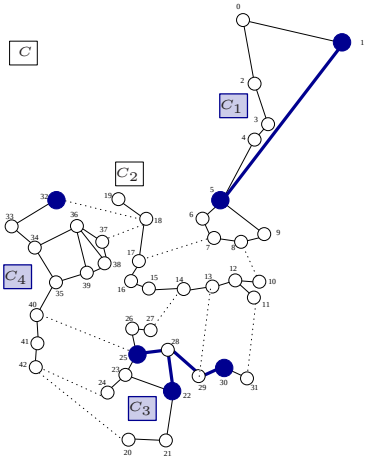
`Topology of the sub-domain` i	$D_i = (V_i, E_i)$
`Groups-labels`	$g_1 \rightarrow l_1$
	...
`Members of the groups`	$g_1 \rightarrow \{b_2, b_3, b_4\}$
	...
`Tree set` $\mathcal{T}$	$l_1 \rightarrow \{\text{edges of } t_1\}$
	...

1. b_3 receives a `join` request for the group g_1.
2. b_3 sends a request to C_i.
3. C_i updates the members of the group g_1 and applies the tree aggregation algorithm to find a tree for g_1. Then, it updates its groups-labels table.
4. C_i informs the routers attached to members of g_1 of the label found.

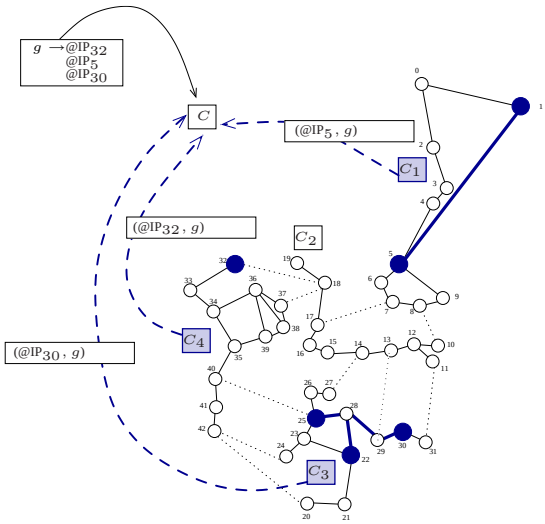
Figure 4. The tree protocol in the sub-domain i managed by C_i.



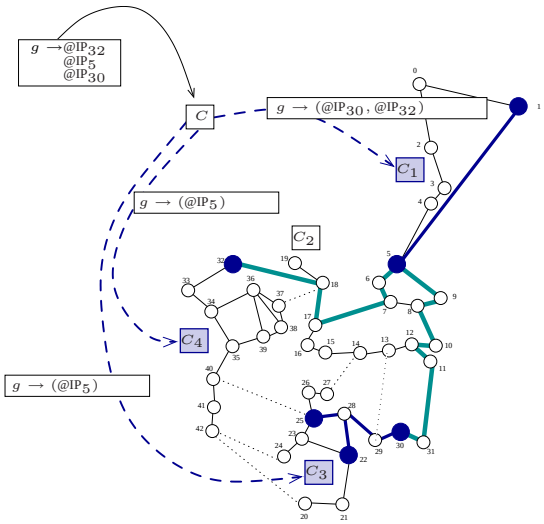
(1) The group g has members in sub-domains 1, 3 et 4.



(2) C_1, C_3 et C_4 aggregate the group g and match it to a set of sub-trees.



(3) Each C_i informs C that some members are present in its sub-domain and gives it the IP address of one of the router attached to members of g.



(4) C sends informations to the C_i in order to configure the tunnels.

Figure 5. The protocol TALD configures firstly a set of sub-trees separately in each of the sub-domain and then tries to connect them by tunnels.

C. Routing in the whole domain

The centralized entities C_i having members of g in their sub-domains have use the algorithm described in previous subsection. In order to route packets for the whole multicast group, the trees in all the sub-domains have to be connected. The centralized entity C, responsible of the main domain D is in charge of this task. Note that C does not need to know the topology of D in order to achieve this. We present in this paper a simple solution to connect these trees in order to validate first the main idea of our proposition.

In this simple solution, each C_i, having members of g in its sub-domain i, has communicated to C the IP address of one of the routers of the sub-domain attached to members of g. This router is the representative router for g in D_i. The centralized entity C keeps this information and maintains the list of the representatives of g for each sub-domain. Note that C does not keep any information concerning the group memberships. Then, C connect the trees in the sub-domains by adding tunnels which can be configured by adding group specific entries matching g to routers in the others sub-domains.

Let us describe an example on Figure 5. Suppose that there are members of group g in sub-domains 1, 3 and 4. Indeed, `join` messages $(g, @IP_1)$ and $(g, @IP_5)$ have been received by C_1, `join` messages $(g, @IP_{22})$, $(g, @IP_{25})$ and $(g, @IP_{30})$ by C_3, and a `join` message $(g, @IP_{32})$ by C_4. Each entity C_i aggregates the sub-group in its sub-domain and informs C that the sub-tree corresponding to group g can be connected through a given node. For example, C_1 informs C that the sub-tree corresponding to g can be connected through node of address $@IP_5$. Now, C has to connect together the three sub-trees corresponding to group g in the three sub-domains. To achieve this connection, C adds two group specific entries $g \rightarrow @IP_{30}$ and $g \rightarrow @IP_{32}$ in

router $@IP_5$. In this way, the three trees in the three sub-domains are connected by tunnels and messages for g can be routed. Note that each entity C_i stores for g the identity of the border routers to which the sub-tree is connected to. Since the sub-tree of C_1 is connected to routers $@IP_{30}$ and $@IP_{32}$, C_1 stores the following entry $g \rightarrow @IP_{30}, @IP_{32})$. Similarly, C_3 stores the entry $grightarrow(@IP_5)$. Notice that this information is only stored for sub-domains that are directly connected to C_i: C_3 does not store any information concerning C_4. At the end, two tunnels are configured: the tunnel $@IP_5 - @IP_{30}$ and the tunnel $@IP_5 - @IP32$.

When routing the messages for a group, the packets are first encapsulated with the label of the tree of the sub-domain. Then, the extremity of the tunnel decapsulates the packet and puts in it the address of the other extremity of the tunnel in order to route the packet towards the other sub-domain.

As our concerns in this paper is to reduce the number of entries stored, we do not optimize the connection of the trees. This can be done as further part of investigation. What only matters for the moment is the number of group specific entries added. If three C_i have registered members of g to C, four group specific entries are added. More generally, if n C_i have replied to C, then $2(n-1)$ entries are needed.

III. Simulations

We run several simulations on different topologies. Due to lack of space, we present only the results of the simulations on the Rocketfuel graph Exodus[2]. This network contains 201 routers and 434 links. During the simulations, 101 routers were core routers and 100 others routers were border routers and can be attached to members of multicast groups. The plots are the results of 100 cases of simulations where each case corresponds to a different set of border routers.

We present the results of the protocols TALD-1, TALD-2 and TALD-4 for different bandwidth thresholds: when 0% bandwidth is allowed to be wasted and when 20% of bandwidth wasted.

1) The protocol TALD-1 represents the actual tree aggregation protocols when the domain is not divided and when aggregation is performed in the main domain.
2) With TALD-2, the domain in divided into 2 sub-domains.
3) With TALD-4, the domain is divided into 4 sub-domains.

The division of the domain was performed by the algorithm presented in Section II-A.

The number of multicast concurrent groups varied from 1 to 10 000 and the number of members of groups was randomly chosen between 2 and 20. The members of groups were chosen randomly among the 100 border routers. This behavior is not representative of the reality but it allows to show the performance of the algorithms in worst-case simulation. Indeed, when the members are randomly located, then the aggregation is more difficult than if members of groups are chosen with some affinity model.

[2] http://www.cs.washington.edu/research/networking/rocketfuel/

A. Number of forwarding states

Figure 6 plots the total number of forwarding states in the domain, *i.e.* the sum of the forwarding states stored by all the routers of the domain. Recall that for a bidirectional tree t, $|t|$ forwarding states have to be stored where $|t|$ denotes the number of routers covered by t. With TALD-1, there is almost no aggregation (the number of forwarding states is the same as if no aggregation was performed) and then, the number of multicast forwarding states is the same with 0% and with 20% of bandwidth wasted. The protocol TALD-4 gives significantly better results than TALD-1 and TALD-2. Moreover, with TALD-4, the number of multicast forwarding states is reduced when the bandwidth threshold is equal to 20%. Finally, the protocol TALD-4 achieves a reduction of 41% of the number of forwarding states compared to TALD-1 when no bandwidth is wasted. This reduction reaches 52% when 20% of bandwidth is wasted.

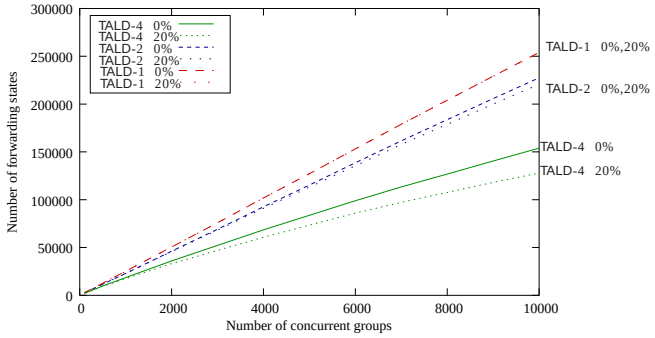


Figure 6. Number of forwarding states

For example, TALD-4 stores around 160 000 forwarding states in the whole domain when the bandwidth threshold is equal to 0% for 10 000 concurrent groups. There is a reduction of 22% when the bandwidth threshold is equal to 20%: the number of forwarding states reaches approximately 126 000. Oppositely, the amount of bandwidth wasted has no influence for the results of TALD-1 as the number of forwarding states is the same when 0% of bandwidth is wasted and when 20% of bandwidth is wasted. This shows that traditional aggregation algorithms are not efficient in large domains.

B. Group specific entries

Figure 7 plots the number of group specific entries which are stored in the group-label table and which match groups to the labels of the aggregated trees. As this number is related to the number of groups, it is not dependent of the bandwidth thresholds and the results

are equivalent for 0% and for 20% of bandwidth wasted. The protocols TALD-2 and TALD-4 need to store more group specific entries in order to route the packets for the groups between the sub-domains. These entries are stored in order to configure the tunnels crossing the sub-domains. Consequently, TALD-1 does not store such entries.

The results show that TALD-4 needs to store more entries than TALD-2 which in turn stores more entries than TALD-1. This is the price to be paid to achieve aggregation and to reduce the number of forwarding states. Note that the more sub-domains, the larger the number of groups specific entries.

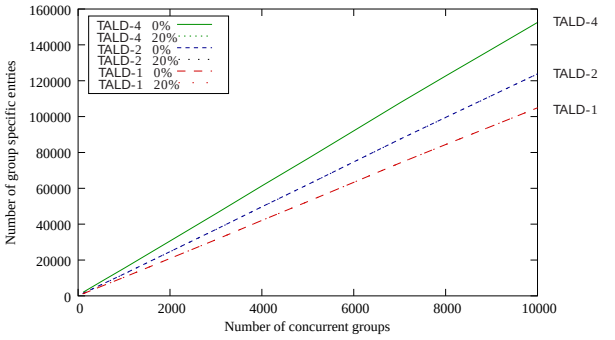


Figure 7. Group specific entries

However, TALD-4 reduces the total number of entries stored in routers compared to TALD-1. Figure 8 shows the total number of the groups specific entries and the forwarding states stored in all the routers of the domain. TALD-4 achieves a reduction of 16% of this total number compared to TALD-1 when no bandwidth is wasted and a reduction of 25% with 20% of bandwidth wasted. It may be noted that TALD-2 does not achieved significant reduction of this number compared to TALD-1. Consequently, dividing the domain in two sub-domains is not enough. However, the memory in routers is significantly reduced with TALD-4. As the number of group specific entries increases with the number of sub-domains, it is not be interesting to divide more the domain. Indeed, the more the domain is divided, the less number of forwarding states but the more the number of group specific entries. Consequently, it may not be interesting to divide the domain into too many sub-domains because the reduction of forwarding states will not be so significant.

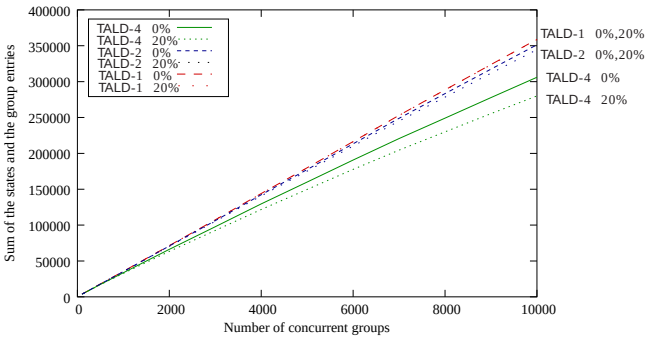


Figure 8. Sum of the forwarding states and of the group specific entries.

C. Mean cost of the trees

Figure 9 shows the mean cost of trees per group. Recall that for TALD-2 and TALD-4, the cost of trees per group represents the cost of all the sub-trees for the group and the cost of the tunnels connecting the sub-trees. That explains why with TALD-2 the mean cost of trees per group is higher than with TALD-1. With TALD-4 the mean cost of trees per group is higher than with TALD-2 and TALD-1. In our simulations, we can see that TALD-4 uses approximately 32 links per group while TALD-2 uses 27 links and TALD-1 uses only only 24 links. This extra-cost for TALD-4 and TALD-2 comparing to TALD-1 is the price to be paid in order to achieve aggregation in large domains. Considering better extremities of the tunnel will allow to spare the resources of the network and to build smaller structures per group. Indeed, the extremities of the tunnels are currently chosen randomly and this behavior is not favorable for the cost of the trees.

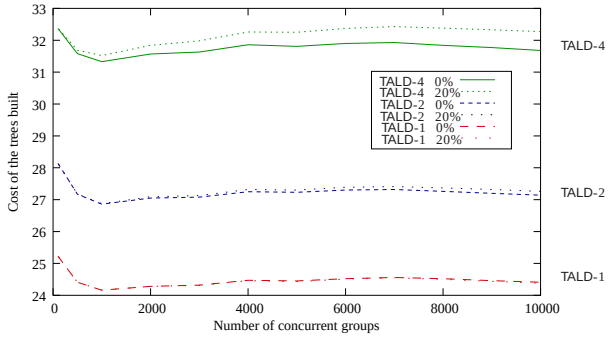


Figure 9. Mean cost of the trees per group

D. Aggregation ratio

Figure 10 shows the aggregation ratio in function of the number of concurrent groups. The aggregation ratio is denoted by the number of trees with aggregation out of the number of trees if no aggregation is performed. Note that for TALD-2 and TALD-4, the number of trees is the sum of the number of trees for each sub-domain.

The protocol TALD-1 achieves less than 1% of aggregation even when 20% of bandwidth is allowed to be wasted. The protocol TALD-4 achieves more than 40% of aggregation even when no bandwidth is allowed to be wasted. When 20% of bandwidth is wasted, the aggregation ratio reaches more than 55%. This figure shows that with large networks, existing algorithms achieving tree aggregation without any division of the domain (as TALD-1) do not realize any aggregation at all.

Figure 11 plots the aggregation ratio in function of the number of border routers in the domain when there are 10 000 concurrent groups. We vary the number of possible border routers among all the 201 routers of Exodus network from 10 to 200. We run 100 times the algorithm for each possible value of the number of border routers in order to get different sets of border routers. The routers that were not border routers could not be attached to members of multicast groups.

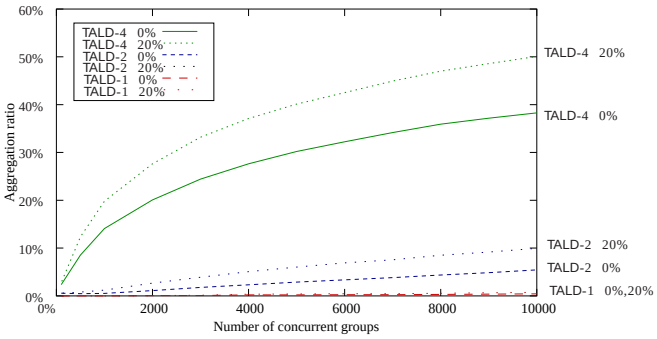


Figure 10. Aggregation ratio

With domains of 10 border routers, the aggregation is very efficient and after 10 000 concurrent groups, the protocols are able to aggregate any new group in the domain. The aggregation ratio decreases dramatically, especially for TALD-1 which is not able to perform any aggregation when the domain contains more than 40 border routers. However, TALD-4 is efficient and performs more than 20% of aggregation even when there are 200 border routers. This shows that for a domain of 40 border routers or more, it is strongly recommended to divide the domain into several sub-domains in order to aggregate groups.

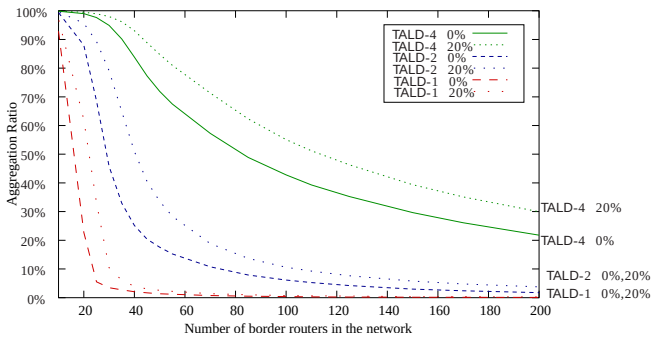


Figure 11. Aggregation ratio in function of the number of border routers

E. Summary of the simulation results

We compared in our simulations the protocols TALD-1, TALD-2 and TALD-4. Recall that TALD-1 can be seen as a traditional aggregation algorithm. The results showed that for the domain Exodus with 100 border routers, TALD-4 was more efficient than TALD-2 in terms of number of forwarding states and aggregation ratio. Moreover, the results showed that without any division, the tree aggregation protocol achieves almost no aggregation at all and behaves in the same way as traditional IP multicast. The price to be paid for this reduction of the number of forwarding states is that the mean cost of trees per group is higher. This is mainly because of the tunnels that are not built in the more efficient way for the moment. Indeed, the tunnels may use links already used by the trees, or by other tunnels. Therefore, some links may be crossed several times for a group: once by the tree and one or more by the tunnels. This behavior can be avoided by choosing in a more efficient way the extremities of the tunnels.

IV. Discussion

Since the goal of this work is to show the feasibility of the tree aggregation in large domains, we left several important issues for future work. The issues concern the way the domain is splitted into several sub-domains (Subsection IV-A and IV-B) and the way the trees in the sub-domains are connected with each other (Subsection IV-C and IV-D).

A. Different algorithms to split a domain

We proposed an algorithm that splits the domain into several sub-domains of roughly the same size. This algorithm is an heuristic in the sense that the sub-domains generated can have different sizes (in the worst-case), even if it is possible to find a set of sub-domains with exactly the same size. However, we are not sure that the size of each sub-domain (or even the number of border routers of each sub-domain) is the right metric. It seems that the important criterion is the total number of different trees in the union of the sub-domains. In order to achieve optimality, two aspects of the problem (namely domain splitting and tree aggregation) have to be considered simultaneously.

B. Optimal number of sub-domains

We showed in our simulations that the performance of our algorithm depends on the number of sub-domains generated. In our experiments, having four sub-domains seem always better than having only one or two. When the number of sub-domains is too small, tree aggregation within each sub-domain is not efficient. On the other hand, having too many sub-domains is not efficient either, since it requires prohibitive amount of control and management for the the sub-domains to be connected together. Moreover, it requires a large number of group specific entries in order to configure the tunnels. This problem is strongly related to the number of border routers in the main domain and the number of border routers needed per sub-domain. Since the optimal number of domains is needed only once for a domain, there is room for a sophisticated, off-line algorithm.

C. Extremities of the tunnels

In the protocol we described, sub-trees are connected together through nodes chosen randomly among the members of the group. One possible solution to achieve this connection is to assign to a set of nodes the function of extremity of the tunnels. That means that only few nodes may be extremities of the tunnels and not all the routers of the sub-domain as done now. Several nodes may be chosen per sub-domain (these nodes can be located in the border of the sub-domain) and when a

tunnel is established the extremities are chosen among these nodes. This behavior allows to put the complexity of encapsulating and decapsulating the packets only at certain nodes. Moreover, only these specific nodes play the role of border routers and not all the routers of the sub-domain.

D. Connecting sub-domains together

Inter-domain protocols have to be aware of different policies to route packets between different intra-domains. Currently, we use unicast tunnels to connect intra-domains together. With few modifications to our protocol, these unicast tunnels could be created according to any inter-domain policy. However, scalability issues might raise again if the number of groups increases. Multicast tunnels seem to be a more viable solution, but comes with management issues such as how to aggregate multicast tunnels together, or how to build policy-aware multicast tunnels.

V. Related Work

We presented in this paper, a tree aggregation protocol specific to large domains. Tree aggregation protocol reduces the number of multicast forwarding states in routers by allowing several groups to share the same delivery tree. In this section, we describe briefly some other propositions that deal with this problem of multicast scalability. Then, we give an overview of the protocols achieving tree aggregation considering different constraints.

A. Reducing the number of multicast forwarding states

The problem of multicast forwarding state scalability has been studied in the literature. Some studies may be found in [3], [4] or in [5].

State Aggregation. State aggregation is achieved either per router [6] or per interface [7]. In [6], the authors propose to aggregate two entries that are successive and that have the same list of output interfaces. Only one entry is stored with the longest common prefix. In this paper, leaky aggregation can be done where, several entries can be aggregated even if they do not share exactly the same list of output interfaces. The union of the lists of output interfaces will be written in the entry and the router forwards some packets for the groups on interfaces that do not lead to members : this is leaky aggregation. The next-hop router that receives the unwanted packet destroys it and some bandwidth is wasted. The authors in [7] propose to place a filter on each output interface. The filter takes in input the address of the group, the output interface and tells if the packet has to be forwarded on the interface or not. When a packet for a group arrives, the filter is applied on each interface in order to forward the packet. With this proposition, the authors reduce the size of the forwarding table by a factor of four.

Explicit Multicast. The protocol Xcast (Explicit Multicast) [8] deals with small group multicast and proposes a new model to achieve the communication. The main idea is that the source puts in the packet the list of the IP addresses of the members when sending the packet. The packet is not forwarded according to the IP multicast address anymore. Therefore, the routers do not maintain any multicast forwarding state. This gain is balanced by an heavy load for the routers that have to manage the Xcast packet containing the list of the receivers and to split the packet if it is a branching router. In this case, the list of the receivers is splitted in sub-lists in the new Xcast packets.

Recursive Unicast Trees. The protocols REUNITE [9] (*Recursive-Unicast approach to multicast*), HBH [10] (*Hop-By-Hop multicast routing protocol*) and SEM [11] (*Simple Explicit Multicast*) propose to store the multicast forwarding states only in the branching routers. The source sends the packets for the group in unicast directly to the next branching router. As the multicast trees contain few branching routers in general, the number of forwarding states to be stored is reduced.

B. Tree Aggregation Protocols

Tree aggregation idea was first proposed in [12] and since, several propositions have been written.

The protocol AM [13], [14] performs aggregation using a centralized entity called the *tree manager* responsible of assigning labels to groups. The protocol STA [15] proposes to speed up the aggregation algorithm with a fast selection function and an efficient sorting of the trees. These two protocols are represented by TALD-1 during the simulations. TOMA [16] is a recent protocol that performs tree aggregation in overlay networks.

Distributed tree aggregation. The distributed protocol BEAM proposed in [17] configures several routers to take in charge the aggregation in order to distribute the work load of the *tree manager*. Indeed, in AM or in STA, only the *tree manager* takes this responsibility. The protocol DMTA [18] proposes to distribute the task of the tree manager among the border routers and then to suppress completely the requests to centralized entities necessary in BEAM to achieve aggregation. In order to propose this distributed protocol, an analysis of the number of trees needed to be configured in a domain is given in [19] and with more details in [20].

Tree aggregation with bandwidth constraints. AQoSM [21] and Q-STA [22] achieve tree aggregation in case of bandwidth constraints. In these two algorithms, links have limited bandwidth capacities and the groups have bandwidth requirements depending on the application they are using. Consequently, groups may be refused if no tree can be built satisfying the bandwidth requirements. While AQoSM tests several source for the native trees in order to builds one that can accept

the group, Q-STA builds native tree maximizing the bandwidth available on the links in order to achieve load balancing and to use in priority the links that are not heavily loaded.

Tree aggregation with tree splitting. The protocol AMBTS [23] performs tree splitting before aggregating groups in order to manage larger domains. A tree is divided into several sub-trees and whenever a new group arrives the native tree is splitted in sub-trees according to a foreclosing process. From these sub-trees, the *tree manager* tries to find already existing sub-trees and to aggregate the group. The idea of AMBTS is somehow orthogonal to the idea of TALD. However, we did not compare AMBTS to TALD during the simulations because of the following reasons.

First of all, the protocol is not realistic for large domains as a centralized entity is responsible of all the process of aggregation. This centralized entity keeps the group memberships for all the groups of the whole domain. Moreover, it is in charge of splitting the trees and aggregating the groups. This behavior is not scalable in domains such as Exodus network with 200 routers. Indeed, too much memory is used to store all the information and the centralized entity is strongly solicited each time a member of a group changes. Second, the foreclosing process in which a tree is divided into several sub-trees is not detailed and we were not able to simulate this algorithm due to lack of information. Splitting the trees manually was not possible in our domain. Finally, the number of sub-trees grows tremendously and is larger than the number of groups (especially if the trees are splitted in many sub-trees). Thus, the process of aggregation is strongly slowed down due to the large number of evaluations of sub-trees. In AMBTS, the simulations were done on a network with 16 border routers. All these reasons make us decide to propose and detail a protocol adequate to large domains.

VI. Conclusion

In this article, we expressed the need of a tree aggregation protocol that can cope with large domains. Indeed, in such cases, traditional tree aggregation protocols do not achieve any significant reduction of the number of forwarding states and behave as traditional IP multicast protocol. We proposed a hierarchical protocol called TALD: the domain is divided into several sub-domains with a centralized entity in each. Each entity aggregates the groups in its sub-domain. There is also a global entity that interconnects the trees together. With such a mechanism, our simulations demonstrate that an aggregation ratio of 20% is attainable in a domain of 200 border routers (while previous tree aggregation protocols achieve an aggregation of ratio of less than 1% in domains of 25 border routers).

Out future work will focus on (i) the splitting of the domain and (ii) the interconnection between the sub-domains.

First, the splitting of the domain can be accomplished in a distributed manner where the routers of the domain realise the splitting themselves. Moreover, the splitting can be done differently and a study of the impact of this splitting on the aggregation ratio may be interesting.

Second, the interconnection of the sub-trees can be achieved in different ways. Presently, it is done by configuring tunnels however, this connection can be achieved by a tree. Moreover, if the connection is still accomplished with tunnels, the selection of the extremities of the tunnels can be optimized. Indeed, the extremities can be chosen among a set of pre-determined extremities located at the border of the domain. In this way, the tunnels have a lower cost.

References

[1] J. Moulierac, A. Guitton, and M. Molnár, "Multicast Tree Aggregation in Large Domains," in *IFIP Networking*, 2006, pp. 691–702.

[2] Eurorings network, "http://www.cybergeography.org/-atlas/kpnqwest_large.jpg ."

[3] M. Sola, M. Ohta, and T. Maeno, "Scalability of internet multicast protocols," in *INET*, 1998.

[4] T. Wong and R. Ratz, "An Analysis of Multicast Forwarding State Scalability," in *International Conference on Network Protocols (ICNP)*, 2000.

[5] B. Zhang and H. Mouftah, "Forwarding State Scalability for Multicast Provisioning in IP Networks," *IEEE Communications Magazine*, June 2003.

[6] P. Radoslavov, D. Estrin, and R. Govindan, "Exploiting the bandwidth-memory tradeoff in multicast state aggregation," Department of Computer Science, USC, Tech. Rep. 99-697, February 1999.

[7] D. Thaler and M. Handley, "On the Aggregatability of Multicast Forwarding State," in *IEEE INFOCOM*, 2000.

[8] R. Boivie, N. Feldman, and C. Metz, "Small Group Multicast: A new Solution for Multicasting on the Internet," *IEEE Internet Computing*, 2000.

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

[10] L. H. M. K. Costa, S. Fdida, and O. C. M. B. Duarte, "Hop-by-Hop Multicast Routing Protocol," in *ACM SIGCOMM*, August 2001.

[11] A. Boudani and B. Cousin, "An hybrid explicit multicast/unicast recursive approach for multicast routing," *Computer Communications*, 2005.

[12] 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.

[13] J.-H. Cui, J. Kim, D. Maggiorini, K. Boussetta, and M. Gerla, "Aggregated multicast — a comparative study," in *IFIP Networking*, ser. LNCS, no. 2345, May 2002.

[14] ——, "Aggregated Multicast — A Comparative Study," *Special issue of Cluster Computing: The Journal of Networks, Software and Applications*, 2003.

[15] A. Guitton and J. Moulierac, "Scalable Tree Aggregation for Multicast," in *8th International Conference on Telecommunications (ConTEL)*, June 2005, best student paper award.

[16] L. Lao, J.-H. Cui, and M. Gerla, "TOMA: A Viable Solution for Large-Scale Multicast Service Support," in *IFIP Networking*, ser. LNCS, no. 3462, May 2005.

[17] 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.
[18] J. Moulierac and A. Guitton, "Distributed Multicast Tree Aggregation," Inria, Tech. Rep. 5636, July 2005.
[19] J. Moulierac, "On the number of multicast aggregated trees in a domain," in *2nd Student Workshop of IEEE Infocom*, April 2006.
[20] J. Moulierac, A. Guitton, and M. Molnár, "On the number of MPLS LSP using Multicast Tree Aggregation," in *IEEE Globecom*, November 2006.
[21] 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.
[22] J. Moulierac and A. Guitton, "QoS Scalable Tree Aggregation," in *IFIP Networking*, ser. LNCS, no. 3462, May 2005.
[23] Z.-F. Liu, W.-H. Dou, and Y.-J. Liu, "AMBTS: A Scheme of Aggregated Multicast Based on Tree Splitting," in *IFIP Networking*, ser. LNCS, no. 3042, May 2004.

Joanna Moulierac obtained her M.Sc. degree in Computer Science at the University of Montpellier II, France in 2003. She is currently a PhD candidate at University of Rennes 1, working with the ARMOR team in IRISA. Her main research interests include active monitoring of networks, group communication with Quality of Service, multicast tree aggregation and deployment of multicast.

Dr Alexandre Guitton obtained his B.Sc. degree at the University of Rouen, France, his M.Sc. degree and his Ph.D. in 2005 at the University of Rennes I, France. He is currently working as a post-doctoral research assistant at Birkbeck College, University of London, United Kingdom. His research focuses on optimization in computer networks, with a particular interest on wireless sensor networks, multicasting and all-optical networks.

Dr Miklós Molnár is an assistant professor at INSA of Rennes, France. He received his Ph.D in 1992 at University of Rennes 1. His research interests include graph theory, combinatorial optimization, high speed networks, traffic engineering, multicast communication and Steiner problem in networks.