Aggregated Multicast – A Comparative Study[⋆]

Jun-Hong Cui, Jinkyu Kim, Dario Maggiorini, Khaled Boussetta, and Mario Gerla

Computer Science Department, University of California, Los Angeles, CA 90095

Abstract. Multicast state scalability is among the critical issues which delay the deployment of IP multicast. In our previous work, we proposed a scheme, called aggregated multicast to reduce multicast state. The key idea is that multiple groups are forced to share a single delivery tree. We presented some initial results to show that multicast state can be reduced. In this paper, we develop a more quantitative assessment of the cost/benefit trade-offs. We introduce metrics to measure multicast state and tree management overhead for multicast schemes. We then compare aggregated multicast with conventional multicast schemes, such as source specific tree scheme and shared tree scheme. Our extensive simulations show that aggregated multicast can achieve significant routing state and tree management overhead reduction while containing the expense of extra resources (bandwidth waste and tunnelling overhead, etc.). We conclude that aggregated multicast is a very cost-effective and promising direction for scalable transit domain multicast provisioning.

1 Introduction

IP Multicast has been a very hot area of research, development and testing for more than one decade since Stephen Deering established the IP multicast model in 1988 [6]. However, IP multicast is still far from being widely deployed in the Internet. Among the issues which delay the deployment, state scalability is one of the most critical ones.

IP multicast utilizes a tree delivery structure on which data packets are duplicated only at fork nodes and are forwarded only once over each link. By doing so IP multicast can scale well to support very large multicast groups. However, a tree delivery structure requires all tree nodes to maintain per-group (or even per-group/source) forwarding information, which increases linearly with the number of groups. Growing number of forwarding state entries means more memory requirement and slower forwarding process since every packet forwarding action involves an address look-up. Thus, multicast scales well to the number of members within a single multicast group. But, it suffers from scalability problems when the number of simultaneous active multicast groups is very large.

To improve multicast state scalability, we proposed a novel scheme to reduce multicast state, which we call *aggregated multicast*. In this scheme, multiple multicast groups are forced to share one distribution tree, which we call an *aggregated tree*. This way, the number of trees in the network may be significantly reduced. Consequently, forwarding state is also reduced: core routers only need to keep state per aggregated tree instead

[⋆] This material is based upon work supported by the National Science Foundation under Grant No. 9805436, and CISCO/CORE fund No. 99-10060 of per group. The trade-off is that this approach may waste extra bandwidth to deliver multicast data to non-group-member nodes. In our earlier work [8,9], we introduced the basic concept of aggregated multicast, proposed an algorithm to assign multicast groups to delivery trees with controllable bandwidth overhead and presented some initial results to show that multicast state can be reduced through inter-group tree sharing. However, a thorough performance evaluation of aggregated multicast is needed: what level of the gain does aggregated multicast offer over conventional multicast schemes? In this paper, we propose metrics to measure multicast state and tree management overhead for multicast schemes. We then compare aggregated multicast with conventional multicast schemes, such as source specific tree scheme and shared tree scheme. Our extensive simulations show that aggregated multicast can achieve significant state and tree management overhead reduction while at reasonable expense (bandwidth waste and tunnelling overhead, etc.).

The rest of this paper is organized as follows. Section 2 gives a classification of multicast schemes. Section 3 reviews the concept of aggregated multicast and presents a new algorithm for group-tree matching. Section 4 then discusses the implementation issues for different multicast schemes and defines metrics to measure multicast state and tree management overhead, and Section 5 provides an extensive simulation study of different multicast schemes. Finally Section 6 summarizes the contributions of our work.

2 A Classification of Multicast Schemes

According to the type of delivery tree, we classify the existing intra-domain multicast routing protocols into two categories (It should be noted that, in this paper, we only consider intra-domain multicasting): in the first category, protocols construct source specific tree, and in the second category, protocols utilize shared tree. For the convenience of discussion, we call the former category as **source specific tree scheme**, and the latter one as **shared tree scheme**. According to this classification, we can say, DVMRP [12], PIM-DM [5], and MOSPF [11] belong to source specific tree scheme category, while CBT [3], PIM-SM [7], and BIDIR-PIM [10] are basically shared tree schemes (of course, PIM-SM can also activate source specific tree when needed).

Source specific tree scheme constructs a separate delivery tree for each source. Namely, each source of a group utilizes its own tree to deliver data to the receivers in the group. The shared tree scheme instead constructs trees based on per-group and all the sources of a group use the same tree to deliver data to the receivers. In other words, multiple sources of the same group share a single delivery tree. Shared tree can be unidirectional or bi-directional. PIM-SM is a unidirectional shared tree scheme. CBT and BIDIR-PIM are bi-directional shared tree schemes. Fig. 1 shows the different types of trees for the same group G with sources $(S1, S2)$ and receivers $(R1, R2)$. For source specific tree schemes, two trees are set up for group G. For the unidirectional shared tree scheme, one tree is set up. Each source needs to unicast packets to the rendezvous point (RP) or build source specific state on all nodes along the path between the source and the RP. For the last scheme, only one bi-directional tree will work. A source can

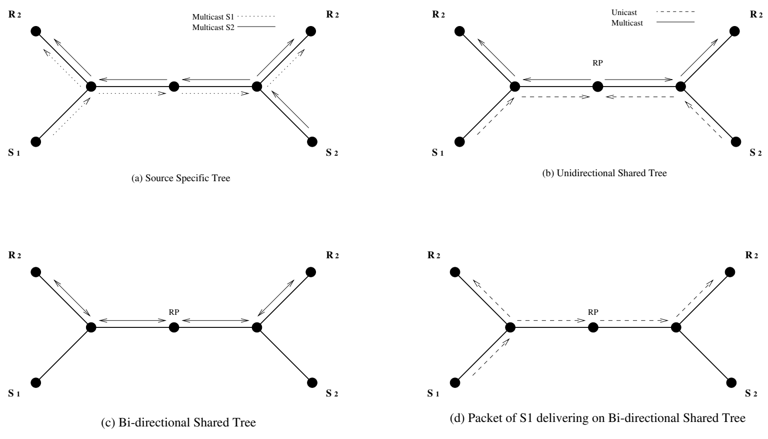
R2
S1
S2
Multicast S1
Multicast S2
(a) Source Specific Tree
Unicast
Multicast
RP
(b) Unidirectional Shared Tree
(c) Bi-directional Shared Tree
(d) Packet of S1 delivering on Bi-directional Shared Tree

Fig. 1. Different types of trees for group G with sources $(S1, S2)$ and receivers $(R1, R2)$.

unicast packet to the nearest on-tree node instead of RP. And each on-tree node can deliver packets along the bi-directional tree.

Compared with conventional multicast schemes, aggregated multicast raises tree-sharing to an even higher level—inter-group tree sharing, where multiple multicast groups are forced to share one aggregated tree. An aggregated tree can be either a source specific tree or a shared tree, while a shared tree can be either unidirectional or bi-directional. We are going to review the basic concept of aggregated multicast and discuss some related issues in the following section.

3 Aggregated Multicast

3.1 Concept of Aggregated Multicast

Aggregated multicast [8,9] is proposed to reduce multicast state, and it is targeted to intra-domain multicast provisioning. The key idea is that, instead of constructing a tree for each individual multicast group in the core network (backbone), multiple multicast groups are forced to share a single aggregated tree.

Fig. 2 illustrates a hierarchical inter-domain network peering. Domain A is a regional or national ISP's backbone network, and domain D, X, and Y are customer networks of

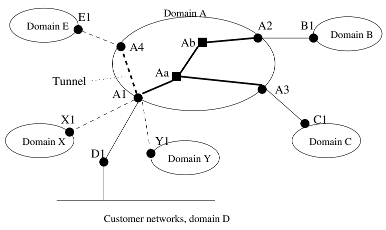

Fig. 2. Domain peering and a cross-domain multicast tree, tree nodes: D1, A1, Aa, Ab, A2, B1, A3, C1, covering group G_0 (D1, B1, C1).

domain A at a certain location (say, Los Angeles), and domain E is a customer network of domain A in another location (say, Seattle). Domain B and C can be other customer networks (say, in Boston) or some other ISP's networks that peer with A. A multicast session originates at domain D and has members in domain B and C. Routers D1, A1, A2, A3, B1 and C1 form the multicast tree at the inter-domain level while A1, A2, A3, Aa and Ab form an intra-domain sub-tree within domain A (there may be other routers involved in domain B and C). Consider a second multicast session that originates at domain D and also has members in domain B and C. For this session, a sub-tree with exactly the same set of nodes will be established to carry its traffic within domain A. Now if there is a third multicast session that originates at domain X and it also has members in domain B and C, then router X1 instead of D1 will be involved, but the sub-tree within domain A still involves the same set of nodes: A1, A2, A3, Aa, and Ab.

To facilitate our discussions, we make the following definitions. For a group G, we call **terminal nodes** the nodes where traffic enters or leaves a domain, A1, A2, and A3 in our example. We call **transit nodes** the tree nodes that are internal to the domain, such as Aa and Ab in our example.

In conventional IP multicast, all the nodes in the above example that are involved within domain A must maintain separate state for each of the three groups individually though their multicast trees are actually of the same "shape". Alternatively, in the aggregated multicast, we can setup a pre-defined tree (or establish a tree on demand) that covers nodes A1, A2 and A3 using a single multicast group address (within domain A). This tree is called an **aggregated tree** (AT) and it is shared by more than one multicast groups (three groups in the above example). We say an aggregated tree T **covers** a group G if all terminal nodes for G are member nodes of T. Data from a specific group is encapsulated at the incoming terminal node using the address of the aggregated tree. It is then distributed over the aggregated tree and decapsulated at exiting terminal nodes to be further distributed to neighboring networks. This way, transit router Aa and Ab only need to maintain a single forwarding entry for the aggregated tree regardless how many groups are sharing it.

Thus, aggregated multicast can reduce the required multicast state. Transit nodes don't need to maintain state for individual groups; instead, they only maintain forwarding state for a smaller number of aggregated trees. The management overhead for the distribution trees is also reduced. First, there are fewer trees that exchange refresh messages. Second, tree maintenance can be a much less frequent process than in conventional multicast, since an aggregated tree has a longer life span.

3.2 Group-Tree Matching in Aggregated Multicast

Aggregated multicast achieves state reduction through inter-group tree sharing—multiple groups share a single aggregated tree. When a group is started, an aggregated tree should be assigned to the group following some rules. If a dense set of aggregated trees is pre-defined, things will be easy: just choose the tree with minimum cost which can cover the group. While in the dynamic case (aggregated tree are established on demand), a more elaborate group-tree matching algorithm is needed.

When we try to match a group G to an aggregated tree T, we have four cases:

1. T can cover G and all the tree leaves are terminal nodes for G, then this match is called **perfect match** for G;
2. T can cover G but some of the tree leaves are not terminal nodes for G, then this match is a **pure-leaky match** (for G);
3. T can not cover G and all the tree leaves are terminal nodes for G, then this match is called a **pure-incomplete match**;
4. T can not cover G and some of the tree leaves are not terminal nodes for G, we name this match as **incomplete leaky match**.

Namely, we denote the case when some of the tree leaves are not terminal nodes for the group G as **leaky match** and the case when the tree can not cover the group G as **incomplete match**. Clearly, leaky match includes case 2 and 4, and incomplete match includes case 3 and 4.

To give examples, the aggregated tree T_0 with nodes (A1, A2, A3, Aa, Ab) in Fig. 2 is a perfect match for our early multicast group G_0 which has members (D1, B1, C1). However, if the above aggregated tree T_0 is also used for group G_1 which only involves member nodes (D1, B1), then it is a pure-leaky match since traffic for G_1 will be delivered to node A3 (and will be discarded there since A3 does not have state for that group). Obviously, the aggregated tree T_0 is an pure-incomplete match for multicast group G_2 which has members (D1, B1, C1, E1) and an incomplete leaky match for multicast group G_3 with members (D1, B1, E1).

We can see that leaky match helps to improve inter-group tree sharing. A disadvantage of leaky match is that some bandwidth is wasted to deliver data to nodes that are not members for the group. Leaky match may be unavoidable since usually it is not possible to establish aggregated trees for all possible group combinations. In the incomplete match case, we have two ways to get a tree for the group. One way is to construct a bigger tree by moving the entire group to a new larger aggregated tree, or, to extend the current aggregated tree to a bigger tree. Extending a tree might involve a lot of overhead, because all the groups which use the extended aggregated tree need to make the corresponding adjustment. An alternative way is to use "tunnelling". Here we give an example. Suppose member E1 in domain E decides to join group G_0 in Fig. 2. Instead of constructing a bigger tree, an extension "tunnel" can be established between edge router A4 (connecting domains A and E) and edge router A1. This solution combines features of multicast inter-group tree sharing and tunnelling; it still preserves core router scalability properties by pushing complexity to edge routers. We can see that, if we employ tunnelling instead of tree extension, then an incomplete match only involves tunnelling. An incomplete leaky match will activate tunnelling and will also waste resources because of leaky matching.

3.3 A New Group-Tree Matching Algorithm

Here we present a new group-tree matching algorithm which is used in our simulation. To avoid the overhead of tree extension, this algorithm uses tunnelling for incomplete match. First, we introduce some notations and definitions.

Overhead Definition. A network is modelled as an undirected graph $G(V, E)$. Each edge (i, j) is assigned a positive cost $c_{ij} = c_{ji}$, which represents the cost to transport a unit of data from node i to node j (or from j to i). Given a multicast tree T, total cost to distribute a unit of data over that tree is

$$C(T) = \sum_{(i,j) \in T} c_{ij}. \tag{1}$$

If every link is assumed to have equal cost 1, tree cost is simply $C(T) = |T| - 1$, where $|T|$ denotes the number of nodes in T. This assumption holds in this paper. Let MTS (Multicast Tree Set) denote the current set of multicast trees established in the network. A "native" multicast tree (constructed by some conventional multicast routing algorithm, denoted by A) for a multicast group G is denoted by T_G^A.

For any aggregated tree T, as mentioned in Section 3.2, it is possible that T does not have a perfect match with group G, which means that the match is leaky match or incomplete match. In leaky match case, some of the leaf nodes of T are not the terminal nodes for G, and then packets reach some destinations that are not interested in receiving them. Thus, there is bandwidth overhead in aggregated multicast. We assume each link has the same bandwidth, and each multicast group has the same bandwidth requirement, then it is easy to get that the percentage bandwidth overhead (denoted by $\delta_L(G,T)$) is actually equal to the percentage link cost overhead:

$$\delta_L(G,T) = \frac{C(T) - C(T_G^A))}{C(T_G^A)}, \tag{2}$$

Apparently, $\delta_L(G,T)$ is 0 for perfect match and pure-incomplete match.

In incomplete match case, T can not cover all the members of group G, and some tunnels need to be set up. Data packets of G exits from the leaf nodes of T, and tunnels to the corresponding terminal nodes of G. Clearly, there is tunnelling overhead caused by unicasting data packets to group terminal nodes. Each tunnel's cost can be measured by the link cost along the tunnel. Assume there are k_G tunnels for group G, and each tunnel is denoted by $T_{G,i}^t$, where $1 \le i \le k_G$, then we define the percentage tunnelling overhead for this incomplete match as

$$\delta_I(G,T) = \frac{\sum_{i=1}^{k_G} C(T_{G,i}^t)}{C(T_G^A)}. \tag{3}$$

It is easy to tell that $\delta_I(G,T)$ is 0 for perfect match and pure-leaky match.

Algorithm Description. Our new group-tree matching algorithm is based on bandwidth overhead and tunnelling overhead. Let l_t be the given bandwidth overhead threshold for leaky match, and t_t be the given tunnelling overhead threshold for incomplete match. When a new group is started,

1. compute a "native" multicast tree T_G^A for G based on the multicast group membership;
2. for each tree T in MTS, compute $\delta_L(G,T)$ and $\delta_I(G,T)$; if $\delta_L(G,T) < l_t$ and $\delta_I(G,T) < t_t$ then T is considered to be a candidate aggregated tree for G;

3. among all candidates, choose the one such that $f(\delta_L(G,T), \delta_I(G,T))$ is minimum and denote it as T_m, then T_m is used to deliver data for G; if T_m can not cover G, the corresponding tunnels will be set up;
4. if no candidate found in step 2, T_G^A is used for G and is added to MTS.

In step 3, $f(\delta_L(G,T), \delta_I(G,T))$ is a function to decide how to choose the final tree from a set of candidates. In our simulations,

$$f(\delta_L(G,T), \delta_I(G,T)) = \delta_L(G,T) + \delta_I(G,T). \tag{4}$$

Actually, this function can be chosen according to the need in the real scenarios. For example, we can give more weight to bandwidth overhead if bandwidth is our main concern.

4 Experiment Methodology

In an aggregated multicast scheme, sharing a multicast tree among multiple groups may significantly reduce the states at network core routers and correspondingly the tree management overhead. However, what level of gain can aggregated multicast get over other multicast schemes? In this section, we will discuss some implementation issues for different multicast schemes in our simulations, and define metrics to measure multicast state and tree management overhead. Then in Section 5, we will compare aggregated multicast with other multicast schemes through simulations.

4.1 Implementation of Multicast Schemes in SENSE

We do our simulations using SENSE (**S**imulation **E**nvironment for **N**etwork **S**ystem **E**volution) [2], which is a network simulator developed at the network research laboratory at UCLA to perform wired network simulation experiments.

In SENSE, we can support the source specific tree scheme, the shared tree scheme (with unidirectional tree and bi-directional tree), and the aggregated multicast scheme (with source specific tree, unidirectional shared tree and bi-directional shared tree). It should be noted that, the multicast schemes we discuss here are not specific multicast routing protocols, since the goal of this paper is to study the gain of aggregated multicast over conventional multicast schemes. The comparison is between schemes, not protocols.

We implement each multicast scheme with a centralized method. For each scheme, there is a centralized processing entity (called *multicast controller*), which has the knowledge of network topology and multicast group membership. The multicast controller is responsible for constructing the multicast tree according to different multicast schemes and then distributing the routing tables to the corresponding nodes. In the implementation, we did not model the membership acquisition and management procedures which depend on the specific multicast routing protocol. This omission reduces the bias and improves the fairness in comparing different multicast schemes. The multicast controller will read group and member dynamics from a pre-generated (or generated on-the-fly) trace file.

For shared tree scheme (either unidirectional or bi-directional) and aggregated multicast scheme with shared tree (unidirectional or bi-directional), a core node or a rendezvous point (RP) is needed when a tree is constructed. To achieve better load balancing, the core node should be chosen carefully. In our implementation, for all multicast schemes using shared trees, a set of possible core routers are pre-configured. Then, when a group is initialized, the core is chosen so as to minimize the cost of the tree.

In an aggregated multicast scheme, the multicast controller also needs to manage aggregated trees and multicast groups and manipulate group-tree matching algorithm. The multicast controller has the same responsibility as the tree manager (mentioned in [8,9]) in aggregated multicast. It collects group join messages and assigns aggregated trees to groups. Once it determines which aggregated tree to use for a group, the tree manager can install corresponding state at the terminal nodes involved.

4.2 Performance Metrics

The main purpose of tree sharing is to reduce multicast state and tree maintenance overhead. So, multicast state and tree management overhead measures are of most concern here. In our experiments, we introduce the following metrics.

Number of multicast trees (or **number of trees** for shorthand) is defined as $|MTS|$, where *MTS* denotes the current set of multicast trees established in the networks. This metric is a direct measurement for the multicast tree maintenance overhead. The more multicast trees, the more memory required and the more processing overhead involved (though the tree maintenance overhead depends on the specific multicast routing protocols).

Forwarding state in transit nodes (or **transit state** for shorthand). Without losing generality, we assume a router needs one state entry per multicast address in its forwarding table. As we defined in Section 3, in a multicast tree, there are transit nodes and terminal nodes. We note that forwarding state in terminal nodes can not be reduced in any multicast scheme. Even in aggregated multicast, the terminal nodes need to maintain the state information for individual groups. So, to assess the state reduction, we measure the forwarding state in transit nodes only.

5 Simulations

In this section, we compare aggregated multicast with conventional multicast schemes through extensive simulation, and quantitatively evaluate the gain of aggregated multicast.

5.1 Multicast Trace Generation

Multicast Group Models. Given the lack of experimental large scale multicast traces, we have chosen to develop membership models that exhibit locality and group correlation preferences. In our simulation, we use the group model previously developed in [9]: **The random node-weighted model**. For completeness, we provide here a summary description of this model.

The random node-weighted model. This model statistically controls the number of groups a node will participate in based on its weight: for two nodes i and j with weight $w(i)$ and $w(j)$ ($0 < w(i), w(j) \leq 1$), let $N(i)$ be the number of groups that have i as a member and $N(j)$ be the number of groups that have j as a member, then it is easy to prove that, in average, $\frac{N(i)}{N(j)} = \frac{w(i)}{w(j)}$. Assuming the number of nodes in the network is N and nodes are numbered from 1 to N. To each node i, $1 \leq i \leq N$, is assigned a weight $w(i)$, $0 \leq w(i) \leq 1$. Then a group can be generated as the following procedure:

```
for $i = 1$ $to$ $N$ do
    generate p, a random number uniformly between 0 and 1, let it be p
    if $p < w(i)$ then
        add i as a group member
    end if
end for
```

Following this model, the average size of multicast groups is $N \sum_{i=1}^{n} w(i)$.

Multicast Membership Dynamics. Generally, there are two methods to control multicast group member dynamics. The first one is to create new members (sources and receivers) for a group according to some pre-defined statistics (arrive rate and member life time etc.), then decide the termination of a group based on the distribution of the group size. This is actually a member-driven dynamic. As to the other method, we call it group-driven dynamics, which means that, group characteristics (group size, group arrival rate, and group life time) are defined first and then group members are generated according to groups. In our experiment, we use the second method, in which the group statistics are controlled first (using the random node weighted model). Actually, the second method looks more reasonable for many real life multicast applications (such as video conference, tele-education, etc.). In any event, the specific method used to control group member dynamics is not expected to affect our simulation results.

In our experiment, given a group life period $[t_1, t_2]$, and the group member set g, where $|g| = n$, for any node $m_i \in g$, $1 \leq i \leq n$, its join time and leave time are denoted by $t_{join}(m_i)$ and $t_{leave}(m_i)$ separately. Then the member dynamics is controlled as follows:

```
for $i = 1$ $to$ $n$ do
    $m_i \in g$
    $t_{join}(m_i)$=get_rand($t_1, t_2$); (get a random time in [$t_1, t_2$])
    $t_{leave}(m_i)$=get_rand($t_{join}(m_i), t_2$); (get a random time in [$t_{join}(m_i), t_2$])
end for
```

It is not difficult to know that the average life time of each member is $|t_2 - t_1|/4$.

5.2 Results and Analysis

We now present results from simulation experiments using a real network topology, vBNS backbone [1].

In vBNS backbone, there are 43 nodes, among which FORE ASX-1000 nodes (16 of them) are assumed to be *core routers* only (i.e. will not be terminal nodes for any multicast group) and are assigned weight 0. Any other node is assigned a weight 0.05 to 0.8 according to link bandwidth of the original backbone router – the rationale is that, the more the bandwidth on the outgoing (and incoming) links of a node, the more the number of multicast groups it may participate in. So, we assign weight 0.8 to nodes with OC-12C links (OC-12C-linked nodes for shorthand), 0.2 to nodes with OC-3C links (OC-3C-linked nodes), and 0.05 to nodes with DS-3 links (DS-3-linked nodes).

In simulation experiments, multicast session requests arrive as a Poisson process with arrival rate λ. Sessions' life time has an exponential distribution with average μ^{-1}. At steady state, the average number of sessions is $\bar{N} = \lambda/\mu$. During the life time of each multicast session, group members are generated dynamically according to group-driven method introduced earlier. Group membership is controlled using the random node-weighted model. Performance data is collected at certain time points (e.g. at $T = 10/\mu$), when steady state is reached, as "snapshot".

First, we design experiments to compare unidirectional shared tree scheme (UST scheme for shorthand) vs aggregated multicast scheme with unidirectional shared tree (AM w/UST scheme for short hand). In this set of experiments, each member of a group can be a source and a receiver. Once a multicast session starts up, its core node (or RP) is randomly chosen from the 16 core routers in the network. For aggregated multicast scheme with unidirectional shared tree, the algorithm specified in Section 3.3 is used to match a group to a tree. When members join or leave a group, its aggregated tree will be adjusted according to the matching algorithm. Correspondingly, the routing algorithm A is PIM-SM like routing algorithm which uses unidirectional shared tree.

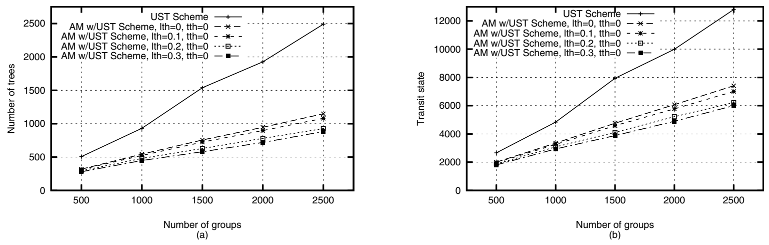

Fig. 3. Results for UST and AM w/UST when only pure-leaky match (tth=0) is allowed

In our first experiment, for aggregated multicast, we only allow pure-leaky match, which means that the tunnelling overhead threshold (represented as **tth**) is 0. We vary the bandwidth overhead threshold (represented as **lth**) from 0 to 0.3. For UST scheme and AM w/UST scheme with different bandwidth threshold, we run simulations to show how the aggregation of aggregated multicast "scales" with the average number of concurrent groups. The results are plotted in Fig. 3. As to the number of trees (see Fig. 3(a)), clearly, for UST scheme, it is almost a linear function of the number of groups. For AM w/UST scheme, as the number of groups becomes bigger, the number of trees also increases, but the increase is much less than UST (even for perfect match ($lth = 0$), the number of trees is only 1150 instead of 2500 for UST when there are 2500 groups). Also this "increase" decreases as there are more groups, which means that as more groups are pumped into the network, more groups can share an aggregated tree. Fig. 3(b) shows us the change of transit state with the number of concurrent groups. It has similar trend to metric number of trees. Transit state is reduced from 12800 to 7400 (above 40% reduction) even for perfect match when 2500 groups come. A general observation is that, when bandwidth overhead threshold is increased, that is, more bandwidth is wasted, number of trees decreases and transit state falls, which means more aggregation. Therefore, there is a trade-off between state and tree management overhead reduction and bandwidth waste.

In our second experiment, for aggregated multicast, we only allow pure-incomplete match, which means that the bandwidth overhead threshold (represented as **lth**) is 0. We vary the tunnelling overhead threshold (represented as **tth**) from 0 to 0.3 and want to look at the effect of tunnelling overhead threshold in the aggregation. Fig. 4 plots the results, which give us curves similar to Fig. 3. However, we can see that tunnelling overhead threshold affects the aggregation significantly: when $tth = 0.3$, and group number is 2500, almost 5 groups share one tree, and transit state is reduced about 70 percentage. When group number increases, we can expect even much more aggregation. The stronger influence of tunnelling overhead threshold on aggregation is not a surprise: the higher the tunnelling overhead threshold is, the more chance for a group to use a small tree for data delivery, the more likely for more groups to share a single aggregated tree.

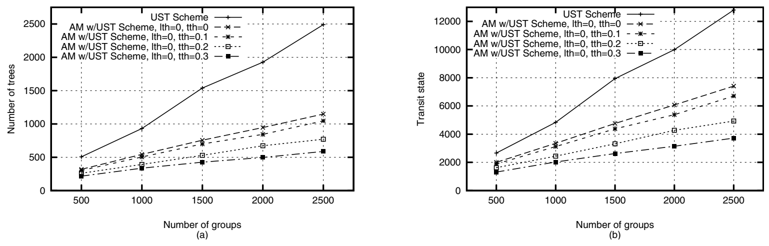

Fig. 4. Results for UST and AM w/UST when only pure-incomplete match (lth=0) is allowed

Our third experiment considers both bandwidth overhead and tunnelling overhead. And the simulation results are shown in Fig. 5. All the results tell what we expect: more aggregation achieved when we sacrifice more (bandwidth and tunnelling) overhead.

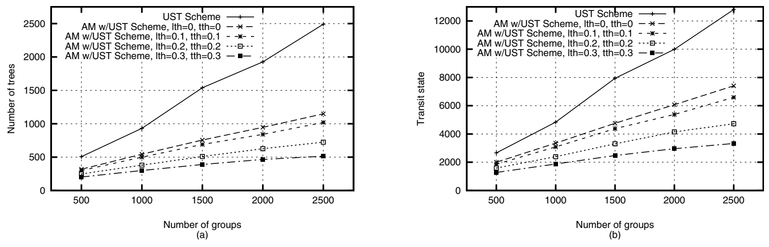

Fig. 5. Results for UST and AM w/UST when both leaky match and incomplete match are allowed

We have shown the results for comparing unidirectional shared tree scheme (UST) vs aggregated multicast scheme with unidirectional shared tree (AM w/UST). Similar results are obtained for source specific tree scheme (SST) vs aggregated multicast scheme with source specific tree (AM w/SST) and bi-directional shared tree scheme (BST) vs aggregated multicast with bi-directional shared tree (AM w/BST). Due to the space limit, we are not going to show the corresponding results for other schemes in this paper. But interested readers can find more results in [4].

From our simulation result and analysis, the benefits of aggregated multicast are mainly in the following two areas: (1) tree management overhead reduction by reducing the number of trees needed to be maintained in the network; (2) state reduction at transit nodes. The price to pay is bandwidth waste and tunnelling cost. The above simulation results confirm our claim while demonstrate the following trends: (1) if we are willing to sacrifice more bandwidth or tunnelling cost (by lifting the bandwidth overhead threshold and tunnelling overhead threshold correspondingly), more or better aggregation is achieved; by "more aggregation" we mean more groups can share an aggregated tree (in average) and correspondingly more state reduction; (2) better aggregation is achievable as the number of concurrent groups increases. The later point is especially important since one basic goal of aggregated multicast is scalability in the number of concurrent groups.

6 Conclusions and Future Work

In this paper, we first gave a classification of multicast schemes, then had a short review of aggregated multicast. For aggregated multicast, we proposed a new group-tree dynamic matching algorithm using tunnelling. We implemented different multicast schemes in SENSE. Through extensive simulations, we compared aggregated multicast with conventional multicast schemes and evaluated its gain over other schemes. Our simulations have shown that significant state and tree management overhead reduction (up to 70% state reduction in our experiments) can be achieved with reasonable bandwidth and tunnelling overhead (0.1 to 0.3), etc.. Thus aggregated multicast is a very promising scheme for transit domain multicast provisioning.

We are now in the process of developing an actual aggregated multicast routing protocol testbed for real application scenarios. The testbed will allow us to better evaluate the state reduction and control overhead.

References

1. vBNS backbone network. *http://www.vbns.net/*.
2. SENSE: Simulation Environment for Network System Evolution. *http://www.cs.ucla.edu/NRL/hpi/resources.html*, 2001.
3. A. Ballardie. Core Based Trees (CBT version 2) multicast routing: protocol specification. *IETF RFC 2189*, September 1997.
4. Jun-Hong Cui, Jinkyu Kim, Dario Maggiorini, Khaled Boussetta, and Mario Gerla. Aggregated Multicast—A Comparative Study. Technical report, UCLA CSD TR No. 020011, February 2002.

5. S. Deering, D. Estrin, D. Farinacci, and V. Jacobson. Protocol Independent Multicast (PIM), Dense Mode Protocol : Specification. *Internet draft*, March 1994.
6. Stephen Deering. Multicast routing in a datagram internetwork. *Ph.D thesis*, December 1991.
7. D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. *IETF RFC 2362*, June 1998.
8. Aiguo Fei, Jun-Hong Cui, Mario Gerla, and Michalis Faloutsos. Aggregated multicast: an approach to reduce multicast state. *In the proceedings of Sixth Global Internet Symposium(GI2001)*, November 2001.
9. Aiguo Fei, Jun-Hong Cui, Mario Gerla, and Michalis Faloutsos. Aggregated Multicast with Inter-Group Tree Sharing. *In the proceedings of NGC2001*, November 2001.
10. Mark Handley and et al. Bi-directional Protocol Independent Multicast (BIDIR-PIM). *Internet draft: draft-ietf-pim-bidir-03.txt*, June 2001.
11. J. Moy. Multicast routing extensions to OSPF. *RFC 1584*, March 1994.
12. C. Partridge, D. Waitzman, and S. Deering. Distance vector multicast routing protocol. *RFC 1075*, 1988.