Aggregated Multicast with Inter-Group Tree Sharing ⋆

Aiguo Fei[1], Junhong Cui[1], Mario Gerla[1], and Michalis Faloutsos[2]

[1] Computer Science Department, University of California, Los Angeles, CA90095
[2] Computer Science and Engineering, University of California, Riverside, CA 92521

Abstract. IP multicast suffers from scalability problems for large numbers of multicast groups, since each router keeps forwarding state proportional to the number of multicast tree passing through it. In this paper, we present and evaluate *aggregated multicast*, an approach to reduce multicast state. In aggregated multicast, multiple groups are forced to share a single delivery tree. At the expense of some bandwidth wastage, this approach can reduce multicast state and tree management overhead at transit routers. It may also simplify and facilitate the provisioning of QoS guarantee for multicast in future aggregated-flow-based QoS networks. We formulate the tree sharing problem and propose a simple intuitive algorithm. We study this algorithm and evaluate the trade-off of aggregation vs. bandwidth overhead using simulations. Simulation results show that significant aggregation is achieved while at the same time bandwidth overhead can be reasonably controlled.

1 Introduction

Multicast is a mechanism to efficiently support multi-point communications. 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. Thus IP multicast is resource-efficient in delivering data to a group of members simultaneously and can scale well to support very large multicast groups. However, even after approximately twenty years of multicast research and engineering effort, IP multicast is still far from being as common-place as the Internet itself.

The deployment of multicasting has been delayed partly because of the scalability issues of the related forwarding state. In unicast, address aggregation coupled with hierarchical address allocation has helped to achieve scalability. This can not be easily done for multicasting, since a multicast address corresponds to a logical group and does not convey any information on the location of its members. A multicast distribution tree requires all tree nodes to maintain per-group (or even per-group/source) forwarding state, and the number of forwarding state entries grows with the number of “passing-by” groups. As multicast gains widespread use and the number of concurrently active groups grows, more and more forwarding state entries will be needed. More forwarding entries translate into more memory requirements, and may also lead to slower forwarding process since every packet forwarding involves an address look-up. This perhaps is the main scalability problem with IP multicast when the number of simultaneous on-going multicast sessions is very large.

⋆ This material is based upon work supported by the National Science Foundation under Grant No. 9805436 and No. 9985195, CISCO/CORE fund No. 99-10060, DARPA award N660001-00-1-8936, TCS Inc., and DIMI matching fund DIM00-10071.

Recognition of the forwarding-state scalability problem has prompted some recent research in forwarding state reduction. Some architectures aim to completely eliminate multicast state at routers [5, 9] using network-transparent multicast, which pushes the complexity to the end-points. Some other schemes attempt to reduce forwarding state by tunneling [11] or by forwarding-state aggregation [8, 10]. Apparently, less entries are needed at a router if multiple forwarding state entries can be aggregated into one. Thaler and Handley analyze the aggregatability of forwarding state in [10] using an input/output filter model of multicast forwarding. Radoslavov et al. propose algorithms to aggregate forwarding state and study the bandwidth-memory tradeoff with simulations in [8]. Both these works attempt to aggregate routing state after the distribution trees have been established.

We propose a novel scheme to reduce multicast state, which we call *aggregated multicast*. The difference with previous approaches is that we force multiple multicast groups 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 of per group. The trade-off is that this approach may waste extra bandwidth to deliver multicast data to non-group-member nodes. Simulation results demonstrate that, the more bandwidth we sacrifice, the more state reduction we can achieve. The management policy and functional requirements can determine the right point in this trade-off. In our earlier work [4], we introduced the basic concepts of aggregated multicast. In this paper, we propose an algorithm to assign multicast groups to delivery trees with controllable bandwidth overhead. We also propose a model to capture the membership patterns of multicast users, which can affect our ability to aggregate groups. Finally, we study the trade-off between aggregation versus bandwidth overhead using series of simulations.

The rest of this paper is organized as follows. Section 2 introduces the concept of aggregated multicast and discusses some related issues. Section 3 then formulates the tree sharing problem and presents an intuitive solution, and Section 4 provides a simulation study of our algorithm and cost/benefit evaluation. Finally Section 5 discusses the implications and contributions of our work.

2 Aggregated Multicast

Aggregated multicast is targeted as an intra-domain multicast provisioning mechanism. The key idea of aggregated multicast is that, instead of constructing a tree for each individual multicast session in the core network (backbone), one can force multiple multicast sessions share a single aggregated tree.

2.1 Concept

Fig. 1 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 domain A at a certain location (say, Los Angeles). Domain B and C can be other customer networks (say, in New York) 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

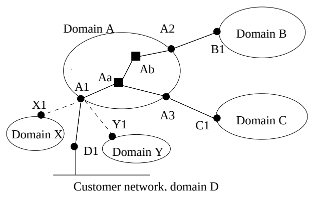

Fig. 1. Domain peering and a cross-domain multicast tree, tree nodes: D1, A1, Aa, Ab, A2, B1, A3, C1, covering group G_0 (D1, B1, C1).

routers involved in domain B and C). The sub-tree can be a PIM-SM shared tree rooted at an RP (Rendezvous Point) router (say, Aa) or a bi-directional shared CBT (Center-Based Tree) tree centered at Aa or maybe an MOSPF tree. Here we will not go into intra-domain multicast routing protocol details, and just assume that the traffic injected into router A1 by router D1 will be distributed over that intra-domain tree and reaches router A2 and A3.

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. 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. Using the terminology commonly used in DiffServ [2], terminal nodes are often *edge* routers and transit nodes are often *core* routers in a network.

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 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. 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. 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. Furthermore, the use of aggregated multicast in one domain is transparent to the rest of the network.

2.2 Discussion

Aggregation reduces the required multicast state in a straightforward way. Transit nodes don't need to maintain state for individual groups; instead, they only maintain forwarding state for a smaller number of aggregated trees. On a backbone network, core nodes are the busiest and often they are transit nodes for many “passing-by” multicast sessions. Relieving these core nodes from per-micro-flow multicast forwarding enables better scalability with the number of concurrent multicast sessions.

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. The control overhead reduction improves the scalability of multicast in an indirect yet important way.

The problem of matching groups to aggregated trees hides several subtleties. The set of the group members and the tree leaves are not always identical. A match is a **perfect** or **non-leaky match** for a group if all the tree leaves are terminal nodes for the group, thus traffic will not “leak” to any nodes that are not group members. For example, the aggregated tree with nodes (A1, A2, A3, Aa, Ab) in Fig. 1 is a perfect match for our early multicast group G_0 which has members (D1, B1, C1). A match may also be a **leaky match**. For example, if the above aggregated tree is also used for group G_1 which only involves member nodes (D1, B1), then it is a 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). A disadvantage of leaky match is that some bandwidth is wasted to deliver data to nodes that are not members for the group (e.g., deliver multicast packets to node A3 in this example). Leaky match may be unavoidable since usually it is not possible to establish aggregated trees for all possible group combinations.

Aggregated multicast can be deployed incrementally and it can interoperate with traditional multicast. First, we can invoke aggregation on a need-to basis. For example, we can aggregate only when the number of groups in a domain goes over a threshold. We can also choose to aggregate only when the aggregation causes reasonable bandwidth overhead, as we will discuss in detail in later sections. Second, aggregated multicast can co-exist with traditional multicast. Finally, the aggregation happens only within a domain, while it is transparent to the rest of the network including neighboring domains.

A related motivation for aggregated multicast is how to simplify the provisioning of multicast with QoS guarantees in future QoS-enabled networks. Regarding QoS support, per-flow-based traffic management requirement of Integrated Services [3] does not scale. Today people are backing away from it and are moving towards aggregated flow based Differentiated Services [2]. The intrinsic per-flow nature of multicast may be problematic for DiffServ networks especially in provisioning multicast with guaranteed service quality. Aggregated multicast can simplify and facilitate QoS management for multicast by pre-assignment of resource/bandwidth (or reservation on demand) in a smaller number of shared aggregated trees.

It is worth pointing out that our approach of “group aggregation” is fundamentally different from the “forwarding-state aggregation” approaches in [8, 10]. We force multiple multicast groups to share a single tree, while their approach is to aggregate multiple multicast forwarding entries on each router locally. In a nutshell, we first aggregate and then route, while they first route and then aggregate. Note that the two approaches can co-exist: it is possible to further reduce multicast state using their approaches even after deploying our approach.

3 The Tree Sharing Problem

To implement aggregated multicast, two main problems must be worked out first: (1) what are the aggregated trees that should be established; (2) which aggregated tree is to be used for a certain group. In this section, we will formulate the tree sharing problem and propose a simple and intuitive algorithm; in the next section, we will present simulation results.

3.1 Why the Problem

If aggregated multicast is only used by an ISP to provide multi-point connections among several routers that have heavy multicast traffic or that are strategically placed to carry inter-domain multicast, then a few number of trees can be pre-established and the matching (from group to a tree) is straightforward. The situation becomes complicated if aggregated multicast is used to a greater extend and the network is large.

Given a network with n edge nodes (nodes that can be terminal nodes for a multicast group), the number of different group combinations is about 2^n (=$C_n^2 + C_n^3 + ... + C_n^n = 2^n - 1 - n$, given that a group has at least two members), which grows exponentially with n. For a reasonably large network, it doesn't make sense to establish pre-defined trees for all possible groups – that number can be larger than the number of possible concurrently active groups. So we should and can only establish a subset of trees out of all possible combinations. This is where **leaky match** comes into play. Meanwhile, if aggregated multicast is used as a general multicast provisioning mechanism, then it becomes a necessity to dynamically manage and maintain aggregated trees since a static set of trees may not be very resource efficient all the time as groups come and go. For any solution one may have, the question is how much aggregation it can achieve and how efficient it is regarding bandwidth use.

3.2 Aggregation Overhead

A network is modeled 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 unit traffic from node i to node j (or from j to i). Given a multicast tree T, total cost to distribute a unit amount of data over that tree is

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

If every link is assumed to have equal cost, tree cost is simply $C(T) = |T| - 1$, where $|T|$ denotes the number of nodes in T.

Given a multicast group g and a tree T, we say tree T **covers** group g if all members of g are *in-tree nodes* of T (i.e., in the vertex set of T). If a group g is covered by a tree T, then any data packets delivered over T will reach all members of g, assuming a bi-directional tree. In a transport network, members of g are not necessarily **group member routers** (i.e., designated router with hosts in its subnet as group members), but rather they are edge routers connecting to other in-tree routers in neighboring domains.

Now consider a network in which routing algorithm A is used to setup multicast trees. Given a multicast group g, let $T_A(g)$ be the multicast tree computed by the routing algorithm. Alternatively, this group can be covered by a aggregated tree $T(g)$, **aggregation overhead** is defined as

$$\varDelta(T, g) = C(T(g)) - C(T_A(g)). \tag{2}$$

Aggregation overhead directly reflects bandwidth waste if tree $T(g)$ is used to carry data for group g instead of the conventional tree $T_A(g)$ with encapsulation overhead not counted; i.e., bandwidth waste can be quantified as $D_g \times \varDelta(T, g)$ if the amount of data transmitted is D_g. Note that, $T_A(g)$ is not necessarily the minimum cost tree (Steiner tree). Therefore, the aggregated tree $T(g)$ may happen to be more efficient than $T_A(g)$, and thus it is possible for $\varDelta(T, g)$ to be negative.

3.3 Two Versions of the Problem

Static Pre-Defined Trees In this version of the problem, we are given: a network $G(V, E)$, tree cost model $C(T)$, a set of N multicast groups, and a number n ($N >> n$). The goal is to find n trees (each of them covers a different node set) and a matching from a group g to a tree $T(g)$ such that every group g is covered by a tree $T(g)$, with the objective of minimizing total aggregation overhead. This is the problem we need to solve to build a set of pre-defined aggregated trees based on long-term traffic measurement information.

In reality, different groups may require different bandwidth and have different life time. Eventually they transmit different amounts of data (to all members, assumed). Aggregation overhead would be $D_g \times \varDelta(T, g)$ for group g which transmits D_g amount of data. However, if D_g is independent of group size and group membership, then statistically the end effect will be the same if all groups are treated as if they have the same amount of data to deliver. Then the total aggregation overhead is simply $\sum_g \varDelta(T, g)$. An **average percentage overhead** can be defined as

$$\delta_A = \frac{\sum_g \varDelta(T, g)}{\sum_g C(T_A(g))} = \frac{\sum_g C(T(g))}{\sum_g C(T_A(g))} - 1. \tag{3}$$

Dynamic Trees The dynamic version of the problem is more meaningful for practical purposes. In this case, instead of a static set of groups, groups dynamically come and go. Our goal is to find a procedure to generate and maintain (establish, modify and tear down) a set of trees and map a group to a tree when the group starts, while minimizing the percentage aggregation overhead.

If an upper bound is put on the number of trees that are allowed simultaneously, apparently the procedure in the dynamic tree matching problem can be used to solve the static tree matching problem: the given set of (static) groups are brought up one by one (without going down) and the dynamic tree matching procedure is used to generate trees and do the mapping; the resulting set of trees and the corresponding mapping are the solution (for the static tree sharing problem).

In the static case, the number of groups given is finite and assumed to be N. In the dynamic case, similarly we can specify N to be the average or maximum number of concurrently active groups. In both the static and dynamic problems, N and n (number of trees allowed) affect aggregation overhead. Intuitively, the closer n to N, the smaller the overhead will be. When $n = N$, the overhead can be 0 since each group can be matched to the tree computed by the routing algorithm. The question is if we can achieve meaningful aggregation ($N >> n$) while bandwidth overhead is reasonable.

3.4 Dynamic Tree Sharing with Aggregation Overhead Threshold Control

Here we present a solution for the dynamic tree sharing problem with the percentage aggregation overhead statistically controlled under a given threshold.

First we introduce some notations and definitions. Let MTS (multicast tree set) denote the current set of multicast trees established in the network. Let $G(T)$ be the current set of active groups covered by tree $T \in MTS$. Both MTS and $G(T)$ evolve with time, and the time parameter is implied but not explicitly indicated. For each $T \in MTS$, an average aggregation overhead for T is kept as

$$\tilde{\delta}(T) = \frac{\sum_{g\ in\ G(T)} \Delta(T,g)}{\sum_{g\ in\ G(T)} C(T_A(g))} = \frac{|G(T)| \times C(T)}{\sum_{g\ in\ G(T)} C(T_A(g))} - 1, \tag{4}$$

and is updated every time $G(T)$ changes. $|G(T)|$ denotes the rank of set $G(T)$(or, the number of groups covered by T). At a certain time t, the average aggregation overhead for all groups $\delta_A(t)$(δ_A at time t) can be computed from $\tilde{\delta}(T,t)$($\tilde{\delta}(T)$ at time t). If tree T is used to cover group g, the percentage aggregation overhead for this match is

$$\delta(T,g) = \frac{C(T) - C(T_A(g))}{C(T_A(g))}. \tag{5}$$

Let $\tilde{\delta}'(T,g)$ denote the average aggregation overhead if group g is covered by T and is to be added to $G(T)$, then

$$\tilde{\delta}'(T,g) = \frac{(|G(T)|+1) \times C(T)}{\sum_{g_x\ in\ \{G(T),g\}} C(T_A(g_x))} - 1 = \frac{(|G(T)|+1) \times C(T)}{\frac{|G(T)| \times C(T)}{1+\tilde{\delta}(T)} + C(T_A(g))} - 1. \tag{6}$$

When a new multicast session goes on with group member set g, there are three options to accommodate this new group: (1) an existing tree covers g and is used to distribute packets for this new session; (2) an existing tree is extended to cover this group; (3) a new tree is established for this group.

Let b_t be the given bandwidth overhead threshold. The goal is to control (statistically) the total percentage overhead to be $\tilde{\delta}(T) < b_t$, for each $T \in MTS$, or $\delta_A < b_t$ (which is a weaker requirement).The following procedure determines how one of the above options is used:

(1) compute the "native" multicast tree $T_A(g)$ for g(e.g., using shortest-path tree algorithm as in MOSPF);

(2) for each tree T in MTS, if T covers g, compute $\tilde{\delta}'(T,g)$; otherwise compute an extended tree T^e to cover g and then compute $\tilde{\delta}'(T^e,g)$; if $\tilde{\delta}'(T,g) < b_t$ or $\tilde{\delta}'(T^e,g) < b_t$, then T or T^e is considered to be a candidate (to cover g);

(3) among all candidates, choose the one such that $C(T)$ or $C(T^e) + |G(T)| \times (C(T^e) - C(T))$ is minimum, denote is as T_m; T_m is used to cover g, update MTS (if T_m is an extended tree), $G(T_m)$, and $\tilde{\delta}(T_m)$;

(4) if no candidate found in step (2), $T_A(g)$ is used to cover g and is added to MTS and correspondingly $G(T_A(g))$ and $\tilde{\delta}(T_A(g))$ are recorded.

To extend tree T to cover group g (step (2)), a greedy strategy similar to Prim's minimum spanning algorithm [6] can be employed to connect T to nodes in g that are not covered, one by one.

Since each group has a limited life time, it will not be using a tree after that. A simple clean-up procedure can be applied when a group goes off: when a multicast session g goes off, g is removed from $G(T)$ where T is the tree used to cover g; if $G(T)$ becomes empty, then T is removed from MTS; T is pruned recursively for nodes no longer needed in the tree; $G(T)$ and $\tilde{\delta}(T)$ are updated. A node is no longer needed in tree T if it is a leaf and is not a member of any group $g \in G(T)$.

In the above algorithm description, we have assumed tree T is a bi-directional tree so that it can be used to cover any group whose members are all in-tree nodes of T. Apparently we can enforce that each tree is source-specific and each group needs to specify a source node, and the above algorithm still applies except that we may turn out to have more trees.

Bandwidth-Aware Aggregation. In all the aggregation overhead definitions we had above, bandwidth requirement of a multicast session is not considered. This is in agreement with today's IP multicast routing architecture where a group's bandwidth requirement is unknown to the routing protocols. At the same time, we assumed both bandwidth requirement and lifetime of a group are independent of the group size and member distribution. If bandwidth requirement is given for each multicast session (e.g., in future networks with QoS guarantee), the above algorithms can be extended to consider the bandwidth in a straightforward way. Due to space limitation, we do not present this formulation here.

3.5 Performance Metrics

We use the following metrics to quantify the effectiveness of an aggregation method.

Let $N(t)$ be the number of active multicast groups in the network at time t and $M(t)$ the number of trees, **aggregation degree** is defined as

$$AD(t) = \frac{N(t)}{M(t)}. \tag{7}$$

AD is an important indication of tree management overhead reduction. For example, the number of trees that need periodical refresh messages to keep state is reduced from N to $\frac{N}{AD}$.

Average aggregation overhead is

$$\delta_A(t) = \frac{\sum_g C(T(g))}{\sum_g C(T_A(g))} - 1 = \frac{\sum_{T \in MTS} |G(T)| \times C(T)}{\sum_{T \in MTS} \frac{|G(T)| \times C(T)}{1+\tilde{\delta}(T)}} - 1, \tag{8}$$

as defined in last subsection. δ_A reflects the extra bandwidth wasted to carry multicast traffic using shared aggregated trees.

Without loss of generality, we assume that a router needs one routing entry per multicast address in its forwarding table. Here we care about the **total number** of state entries that are installed at **all** routers involved to support a multicast group in a network. In conventional multicast, the total number of entries for a group equals the number of nodes $|T|$ in its multicast tree T (or subtree within a domain, to be more specific) – i.e., each tree node needs one entry for this group. In aggregated multicast, there are two types of state entries: entries for the shared aggregated trees and group-specific entries at terminal nodes. The number of entries installed for an aggregated tree T equals the number of tree nodes $|T|$ and these state entries are considered to be **shared** by **all groups** using T. The number of group-specific entries for a group equals the number of its terminal nodes because only these nodes need group-specific state.

Furthermore, we also introduce the concept of **irreducible state** and **reducible state**: group-specific state at terminal nodes is **irreducible**. All terminal nodes need such state information to determine how to forward multicast packets received, no matter in conventional multicast or in aggregated multicast. For example, in our early example illustrated by Fig. 1, node A1 always needs to maintain state for group G_0 so it knows it should forward packets for that group received from D1 to the interface connecting to Aa and forward packets for that group received from Aa to the interface connecting to node D1 (and not X1 or Y1), assuming a bi-directional inter-domain tree.

Given a set of groups $\mathcal{G}$, if each group g is serviced by a tree $T_A(g)$, then the total number of state entries is

$$N_A = \sum_{g \in \mathcal{G}} |T_A(g)|. \tag{9}$$

Alternatively, if the same set of groups are serviced using a set of aggregated trees MTS, the total number of state entries is

$$N_T = \sum_{T \in MTS} |T| + \sum_{g \in \mathcal{G}} |g|, \tag{10}$$

where $|T|$ is the number of nodes in T, and $|g|$ is the group size of g. The first part of (10) represents the number of entries to maintain for the aggregated trees, while the second part denotes the number of entries that source and exit nodes of a group need to maintain in order to determine how to forward and handle multicast data packets. **Overall state reduction ratio** can be defined as

$$r_{as} = 1 - \frac{N_T}{N_A} = 1 - \frac{\sum_{T \in MTS} |T| + \sum_{g \in \mathcal{G}} |g|}{\sum_{g \in \mathcal{G}} |T_A(g)|}. \tag{11}$$

A better reflection of state reduction achieved by our "group aggregation" approach, however, is the **reducible state reduction ratio**, which is defined as

$$r_{rs} = 1 - \frac{\sum_{T \in MTS} |T|}{\sum_{g \in \mathcal{G}} (|T_A(g)| - |g|)}; \tag{12}$$

i.e., the total number of entries needed to be maintained by transit nodes has been reduced from $\sum_{g \in \mathcal{G}} (|T_A(g)| - |g|)$ to $\sum_{T \in MTS} |T|$.

Another metric called **hit ratio** is defined as

$$HR(t) = \frac{number\ of\ groups\ covered\ by\ existing\ trees}{total\ number\ of\ groups} = \frac{N_h(t)}{N_t(t)}. \tag{13}$$

Both N_h and N_t are accumulated from time 0 (start of simulation) to time t. The higher $HR(t)$, the less often new trees have to be setup to accommodate new groups. Similarly **extend ratio** is defined as

$$ER(t) = \frac{number\ of\ groups\ covered\ by\ extended\ trees}{total\ number\ of\ groups} = \frac{N_e(t)}{N_t(t)}. \tag{14}$$

The "cost" to extend an existing is expected to be lower than setting-up a new tree. Percentage of groups that require to establish new trees is $1 - HR - ER$, up to time t.

4 Simulation Studies

In this section, we evaluate our approach by studying the trade-off between aggregation and bandwidth overhead using simulations. We find that we can achieve significant aggregation for reasonable bandwidth overhead. We attempt to test our approach in a wide variety of scenarios. Given the absence of large scale real multicast traces, we are forced to develop membership models that exhibits a locality and correlated group preferences.

4.1 Multicast Group Models

The performance of any aggregation is substantially affected by the distribution of multicast group members in the network. Currently multicast is not widely deployed and its usage has been limited, thus, trace data from real multicast sessions is limited and can only be considered as an indication of multicast patterns in large scale multicast. We develop and use the following different models to generate multicast groups.

In most multicast routing research literature, members of a multicast group are randomly chosen among all nodes. In this model, not only group members are assumed to be uncorrelated, but all nodes are treated the same as well. This well reflects member distribution of many applications, such as Internet gaming, but not all of them. In some applications, members tend to cluster together; for example, an Internet broadcast of Laker's basket ball game is watched by its local fans around Los Angeles area. Inter-group correlation is also an important factor; for example, members of a multicast group might tend to be in another group as well [12]. Neither does this model reflect the fact that not all nodes in the network are equivalent. For example, consider two nodes in MCI's backbone network: one is in Los Angeles and the other one is in Santa Barbara. It is very likely that the LA node has much more multicast sessions going through it than that of the Santa Barbara node given that MCI has a much larger customer base in LA. Besides, there can be historic correlation of group distribution among network nodes as well. For example, a customer company of MCI's Internet service has three locations in LA, Seattle and Houston, which are connected through MCI's backbone. There are often video conferences among these three sites, and when there is one, MCI's routers at the three places will be in a multicast session.

From the above discussions, to model multicast group distribution, the following factors have to be considered: (1) member distribution within a group (e.g., spread or clustered); (2) inter-group correlation; (3) node difference in multicast participation; (4) inter-node correlation; (5) group size distribution; i.e, how often we tend to have very small groups or very large groups. Factor (5) has been not discussed above, but clearly it is very important as well. Several models are described in [12] where factors (1) and (2) are considered. The terms of affinity and disaffinity are used in [7] to describe the clustering and spreading out tendencies of members within a group.

In our work, we use the **node weighted** framework which incorporates the difference among network nodes (factor (3)). In this framework, each node is assigned a weight representing the probability for that node to be in a group. We have two models to generate groups based on node weight assignment, which gives rise to two different models.

The random node-weighted model. This model statistically controls the number of groups a node will participate 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 $N(i)$ =the number of groups that have i as a member and $N(j)$ =the number of groups that have j as a member, then $\frac{N(i)}{N(j)} = \frac{w(i)}{w(j)}$ in average. Assuming the number of nodes in the network is N and nodes are numbered from 1 to N. For each node i, $1 \leq i \leq N$, it 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 a random number between 0 and 1, let it be p
    if $p < w(i)$ then
        add i as a group member
    end if
end for
```

The group-size controlled model. In this model, we want to have more accurate control over the size of the groups we generate. For this reason, we use the following procedure to generate groups with a given group size that follows a given probability mass function pmf $p_X(x)$:

```
generate group size $n$ according to $p_X(x)$
while the number of member is less than $n$ do
    randomly pick up a non-member, let it be i
    generate a random number between 0 and 1, let it be p
    if $p < w(i)$ then
        add i as a group member
    end if
end while
```

This model controls the group-size distribution; however, nodes no longer participate in groups according to their weights (i.e., we no longer have $\frac{N(i)}{N(j)} = \frac{w(i)}{w(j)}$ in average).

4.2 Simulation Results

We present results from simulation using a network topology abstracted from a real network topology, AT&T IP backbone [1], which has a total of 123 nodes: 9 gateway routers, 9 backbone routers, 9 remote GSR (gigabit switch router) access router, and 96 remote access routers.

The abstract topology is constructed as follows. First, we "contract" all the attached remote access routers of a gateway router or a backbone router into one node (connecting to the original gateway/backbone router), which is called a **contracted node**. Since a gateway router in the backbone represents connectivity to other peering network(s) and/or Internet public exchange point(s), a neighbor node called **exchange node** is added to each gateway router to represent such external connectivity. The result is a simplified network with 54 nodes. Among these nodes, gateway nodes (9 of them) and backbone nodes (9 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. Each access router is assigned weight 0.01, and a "contracted" node's weight is the summation of the weights of all access routers from which it is contracted. Exchange nodes are assigned weight ranging from 0.1 to 0.9 in different simulation runs.

In simulation experiments, multicast connection requests arrive as a Poisson process with arrival rate λ. Each time a connection comes up, all group members are specified. Group membership is generated using the node weighted framework discussed in Section 4.1. Connections' life time has a Poisson distribution with average μ. At steady state, average number of connections is $\bar{N} = \lambda \times \mu$. The algorithm specified in last section is used to establish/mantain trees and map a group to a tree. The routing algorithm A is shortest-path tree algorithm with root randomly chosen from all group members. Performance data is collected at certain time points (e.g., at $T = 10\mu$), when stead state is reached, as "snapshot".

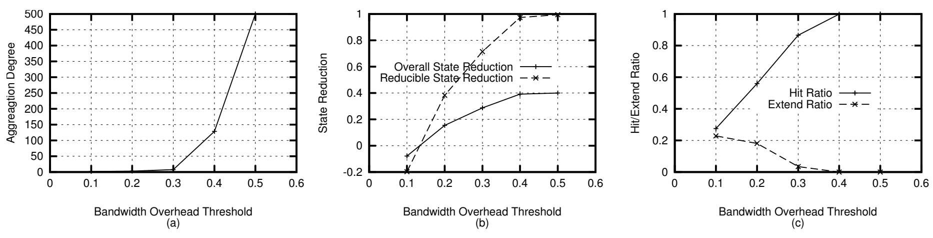

Fig. 2. Aggregation vs. bandwidth overhead threshold, groups are generated using group-size-controlled model.

In our first experiment, an exchange node is assigned a weight 0.25 or 0.9 according to link bandwidths of the original gateway– 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. Groups are generated using the group-size-controlled model. Group size is uniformly distributed from 2 to 36 and the average number of concurrently active groups is $\lambda\mu = 1000$. Fig. 2 shows the results of aggregation degree (a), state reduction (b) and hit/extend ratios (c) vs. bandwidth overhead threshold. We can see that, aggregation degree increases as the bandwidth threshold is increased – if we are willing sacrifice more bandwidth, we can accommodate more multicast groups into a shared aggregated tree. Apparently this agrees with our intuition. Fig. 2(b) shows that, overall state reduction ratio and reducible state ratio also increase with bandwidth overhead threshold – as we squeeze more groups into an aggregated tree, we need fewer trees and achieve more state reduction. Fig. 2(c) tells us that, hit ratio goes up and extend ratio goes down with increasing threshold. This is consistent with the trend of other metrics (aggregation degree and state reductions). When more groups can share an aggregated tree, it is more likely for an incoming group to be covered by an existing tree and thus it becomes less often to setup new trees or extend existing trees.

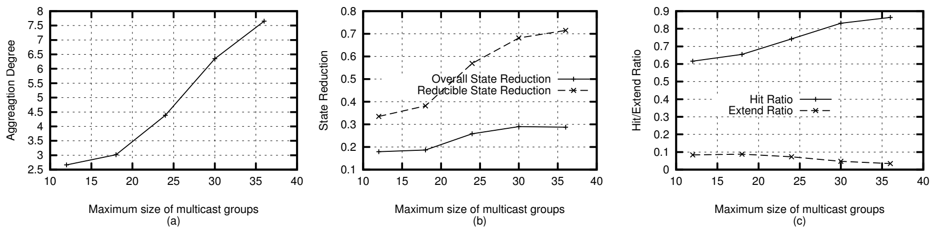

Fig. 3. Aggregation vs. maximum size of multicast groups, groups are generated using group-size-controlled model.

In our second experiment, we keep the bandwidth overhead threshold at a fixed value (=0.3 for results presented here) and vary the upper bound of group size (still uniformly distributed with lower bound 2) while keep all other parameters the same as in the first experiment. The results in Fig. 3 demonstrate the effect of group size on aggregation: if there are more larger groups, then we can aggregate more groups into sharing trees. As groups become larger, so do their multicast trees. A larger tree can "cover" more groups than a smaller one under the same overhead threshold (i.e., there are more subtrees of a larger tree within that threshold).

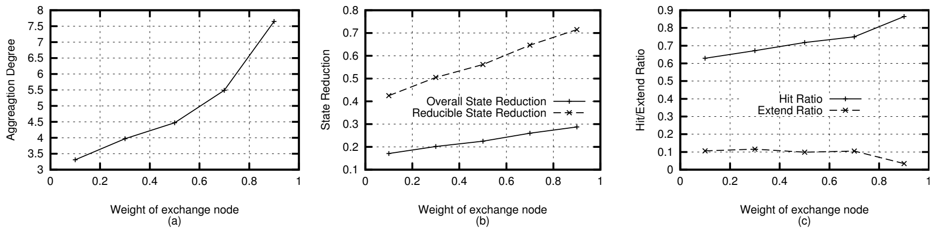

Fig. 4. Aggregation vs. weight of exchange nodes, groups are generated using group-size-controlled model.

We want to see how node weight affects aggregation. Here we also keep the bandwidth overhead threshold at a fixed value (=0.3 for results presented here) and group size is uniformly distributed from 2 to 36. All other parameters are the same as in our first experiment, while we vary the weight of *exchange nodes* from 0.1 to 0.9. The results are shown in Fig. 4. The higher the weight of a node, the larger the number of groups it may participate. As we increase the weights of those *exchange nodes*, multicast groups are more likely to "concentrate" on these nodes, and better aggregation is achieved as the results show.

The same set of experiments are also conducted using the random node-weighted model. In Fig. 5, we plot the results of an experiment similar to our first one. The results demonstrate similar trends, although the actual values differ. Note that the state reduction seems to be comparable.

We also examine how the aggregation scales with the (statistic average) number of concurrent groups. We run simulations with different $\lambda\mu$ products while keep all other

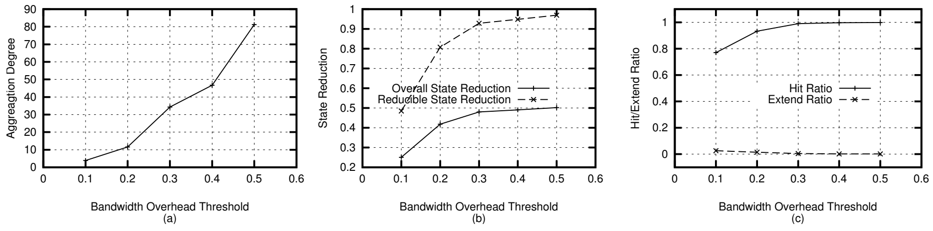

Fig. 5. Aggregation vs. bandwidth overhead threshold, groups are generated using random model.

parameters fixed. Fig. 6 plots the results for two different bandwidth overhead thresholds. It is no surprise that as more groups are pumped into the network, the aggregation degree increases – in average, more groups can share an aggregated tree. The scaling trend is encouraging: as the average number of (concurrent) groups is increased from 1000 to 9000, the number of aggregated trees is increased from 29 to 46 only (with bandwidth overhead threshold 0.3). At the same time, reducible state reduction ratio is getting close to 1.

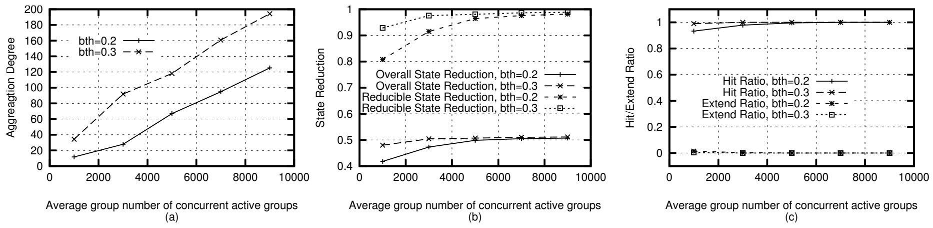

Fig. 6. Aggregation vs. concurrent group number, groups are generated using random model.

To summarize, we observe that the state reduction can be significant up to 50% for the overall state. Furthermore, we can get significant reduction even for small bandwidth overhead. Finally, our approach has the right trend: the state-reduction increases as the number and the size of groups increase. This way, the aggregation becomes more effective when it is really needed.

We also have to warn the limitations of such simulation studies. As we have found, multicast group models (size distribution and member distribution as controlled by node weights) can significantly affect aggregation; thus how aggregated multicast is going to work out in real networks depends a lot on such factors in practice. Therefore, it is important to develop realistic multicast scenarios to evaluate any aggregation approach.

5 Conclusions and Future Work

We propose a novel approach to address the problem of multicast state scalability. The key idea of aggregated multicast is to force groups into sharing a single delivery tree. This comes in contrast to other forwarding-state aggregation approaches that first create multiple trees and then try to aggregate the state locally on each router. A key concept in our approach is that it sacrifices bandwidth to reduce the routing state.

Our work could be summarized in the following points:

– We introduce the concept of aggregated multicast and discuss several related issues.
– We formulate the tree sharing problem and present a simple and effective algorithm to establish aggregated trees and dynamically match groups with existing trees.
– We propose performance metrics that can be used to evaluate our approach.
– We show that our approach seems to be very promising in a series of simulation experiments. We can achieve significant state aggregation (up to 50%) with relatively small bandwidth overhead(10% to 30%).

Our work suggests that the benefits of aggregated multicast lie in the following two areas: (1) control overhead reduction by reducing the number of trees needed to be maintained in the network; (2) state reduction at core nodes. While the price to pay for that is bandwidth waste. Our simulation results confirm our claim while demonstrate the following trends: (1) as we are willing to sacrifice more bandwidth (by increasing the bandwidth overhead threshold), more or better aggregation is achieved; (2) better aggregation is achievable as the number and size of concurrent groups increases. The last is specially important since one basic goal of aggregated multicast is to achieve better scalability regarding the number of concurrent groups.

FUTURE WORK. Our scheme simplifies multicast management and could lend itself to a mechanism of QoS provisioning and multicast traffic engineering with the appropriate use of DiffServ or MPLS. We find that this by itself could be a sufficient motivation for studying aggregated multicast.

References

1. AT&T IP Backbone. *http://www.ipservices.att.com/backbone/*, 2001.
2. S. Blake and et al. An architecture for differentiated services. *IETF RFC 2475*, 1998.
3. R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. *IETF RFC 1633*, 1994.
4. Aiguo Fei, Jun-Hong Cui, Mario Gerla, and Michalis Faloutsos. Aggregated multicast: an approach to reduce multicast state. *To appear in Sixth Global Internet Symposium(GI2001)*, November 2001.
5. P. Francis. Yoid: extending the internet multicast architecture. *http://www.aciri.org/yoid/docs/index.html*.
6. U. Manber. *Introduction to Algorithms: a Creative Approach*. Addison-Wesley Publishing Company, 1989.
7. G. Philips and S. Shenker. Scaling of multicast trees: comments on the chuang-sirbu scaling law. In *Proceedings of ACM SIGCOMM'99*, pages 41–51, September 1999.
8. P. I. Radoslavov, D. Estrin, and R. Govindan. Exploiting the bandwidth-memory tradeoff in multicast state aggregation. Technical report, USC Dept. of CS Technical Report 99-697 (Second Revision), July 1999.
9. Y. Chu S. Rao and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics'00*, June 2000.
10. D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. In *Proceedings of IEEE INFOCOM'00*, Tel Aviv, Israel, March 2000.
11. J. Tian and G. Neufeld. Forwarding state reduction for sparse mode multicast communications. In *Proceedings of IEEE INFOCOM'98*, San Francisco, California, March 1998.
12. T. Wong and R. Katz. An analysis of multicast forwarding state scalability. In *Proceedings of the 8th International Conference on Network Protocols (ICNP)*, Japan, November 2000.