

# Tackling Group-To-Tree Matching in Large Scale Group Communications<sup>\*</sup>

Li Lao<sup>a</sup>, Jun-Hong Cui<sup>b</sup>, Mario Gerla<sup>a</sup>

<sup>a</sup>*Computer Science Department, University of California, Los Angeles*

<sup>b</sup>*Computer Science & Engineering Department, University of Connecticut, Storrs*

---

## Abstract

As a mechanism to efficiently support group communications, multicasting, faces a serious state scalability problem when there are large numbers of groups in the network. Recently, a novel solution called **Aggregated Multicast** has been proposed, in which multiple groups can share one delivery tree. A key problem in Aggregated Multicast is group-to-tree matching (i.e., assigning groups to proper trees). In this paper, we formally define this problem, and formulate two versions of the problem: static and dynamic. We analyze the static version and prove that it is NP-complete. To tackle this hard problem, we propose three algorithms: one optimal (using Linear Integer Programming, or ILP), one near-optimal (using Greedy method), and one pseudo-dynamic algorithm. For the dynamic version, we present a generic dynamic on-line algorithm. Simulation study has been conducted to evaluate the performance of the algorithms. Our results show that: 1) for the static problem, the Greedy algorithm is a feasible solution and its performance is very close to the optimal ILP solution, while the pseudo-dynamic algorithm is a good heuristic for many cases where Greedy does not work well; 2) for the dynamic problem, the generic dynamic on-line algorithm is a very practical solution with promising performance and reasonable computation requirement.

### *Key words:*

multicast, group communication, aggregated multicast, group-to-tree matching

---

<sup>\*</sup> This material is based upon work supported by the National Science Foundation under Grant No. 0435515 and Grant No. 0435230.

*Email addresses:* llao@cs.ucla.edu (Li Lao), jcui@cse.uconn.edu (Jun-Hong Cui), gerla@cs.ucla.edu (Mario Gerla).

## 1 Introduction

With the rapid development of the Internet, there are many emerging large scale multi-user applications, such as distributed interactive simulation (DIS), distributed network games, teleconferencing and telemedicine. All these applications involve multi-point group communications (that is, delivering data from one or more sources to multiple receivers). To support these applications efficiently, multicast is usually employed, in which a concept of group is introduced: sources send data to an advertised group; receivers who are interested in the data need to subscribe to the group in order to receive the data.

In multicast protocols, a tree delivery structure is widely used due to its resource efficiency. Each in-tree node maintains forwarding state, and data packets are duplicated at fork nodes and are forwarded only once over each link. Traditionally, each multicast group uses one delivery tree. To manage multicast groups, resources (e.g., memory to maintain group forwarding state) and control overhead (e.g., setup and maintenance of multicast trees) are required. When there are large numbers of multicast groups in the network, a lot of resources and management overhead will be involved. Hence, the network performance will be tremendously degraded. This issue is referred to as multicast state scalability problem. It will be exacerbated with the increasing demand of multi-user applications.

The recognition of the state scalability problem has prompted a significant amount of interesting work. Some approaches resort to different multicast architectures, such as application-layer multicast [18,9] and Xcast [6], aiming to completely eliminate multicast state at routers. These solutions either sacrifice multicast efficiency or increase packet processing overhead at routers. Some other schemes attempt to reduce forwarding state at non-branching tree nodes [29,27,11,30] or aggregate forwarding state at individual nodes [24,28,8]. However, all of them only consider reducing forwarding state at routers without solving the control overhead issue.

On the other hand, a recently proposed approach called Aggregated Multicast [16] exploits both the resource and control overhead issues. In this scheme, multiple multicast groups are aggregated to share a single delivery tree (which is called an *aggregated tree*). This way, the total number of trees in the network may be significantly reduced and thus the forwarding state would be decreased accordingly. Aggregated multicast involves group-to-tree matching (i.e., assigning groups to trees) since proper trees should be found to deliver data for groups. To solve the state scalability problem, the objective of group-to-tree matching algorithms would be to minimize the resources and control overhead. In previous study [26,14,13], several Aggregated Multicast protocols using heuristic on-line group-to-tree matching algorithms have been proposed;

however, there is no formal analysis of the group-to-tree matching problem, and there is no formal evaluation of how good the on-line heuristics are in comparison with optimal solutions.

In this paper, we formally define the group-to-tree matching problem and formulate two versions of the problem: static and dynamic. In the static version, we assume all the groups are known beforehand, i.e., we have the knowledge of global group information. This case is useful for multicast tree pre-dimensioning based on long-term traffic measurement. We analyze the complexity of this problem and show that it is NP-complete. We propose three algorithms: one optimal (using Linear Integer Programming, or ILP), one near-optimal (using Greedy method), and one pseudo-dynamic algorithm. By simulation study, we show that the Greedy method is much faster and less time consuming than ILP while the performance is not significantly sacrificed. The pseudo-dynamic algorithm is even faster and more resource efficient, but it trades off the performance for efficiency. For the dynamic version of the problem, groups dynamically join and leave, and there is no global information about all the groups. This is more meaningful for managing on-line systems. We present a generic dynamic on-line group-to-tree matching algorithm, and evaluate its performance by comparing it with its upper bound (obtained using the static algorithms) and existing on-line heuristics. We find that the generic dynamic on-line algorithm is a practical solution to the dynamic group-to-tree matching problem with limited performance penalty and reasonable computation requirement.

The rest of this paper is organized as follows. In Section 2, we first describe and formulate the group-to-tree matching problem. Then we present several algorithms for both the static and dynamic versions of the problem, and give formal time complexity analysis of these algorithms (Section 3, 4 and 5). After that, we conduct simulation study and compare different algorithms quantitatively in Section 6. Finally, we discuss some related work in Section 7, and conclude the paper in Section 8.

## 2 The Group-To-Tree Matching Problem

### 2.1 Problem Description

In traditional multicast, each group uses one delivery tree, while in Aggregated Multicast [16], multiple groups are forced to share one aggregated tree. Thus, to implement Aggregated Multicast, we need to conduct group-to-tree matching to assign groups to proper trees. One thing to note is that Aggregated Multicast is targeted for a single domain, especially a transit domain,

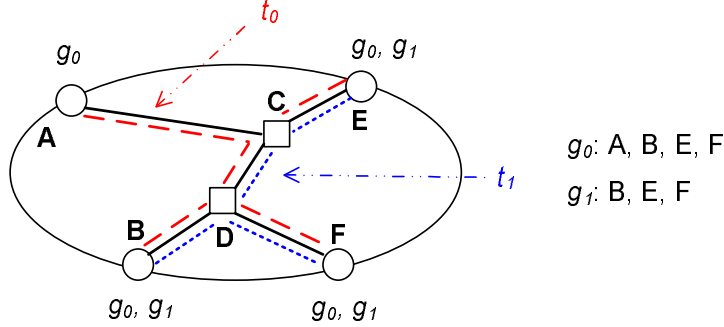


Fig. 1. Examples of Perfect Match and Leaky Match. Tree  $t_0$  is a perfect match for group  $g_0$ ,  $t_1$  is a perfect for  $g_1$ , and  $t_0$  is a leaky match for  $g_1$ .

where member dynamics are usually not as significant as in stub domains [16]. Thus, in this paper, we also consider the group-to-tree matching problem in a single domain.

Given a group and a tree, the group members (sources and receivers) and the tree leaves are not always identical. If all the tree leaves have group members, this tree is called a *perfect match* for the group; otherwise, we call it a *leaky match*. In the latter case, the tree is “bigger” than the group. In other words, we send data to parts of the tree with no interested receivers. Some simple examples of perfect match and leaky match are illustrated in Figure 1. In the target network domain, there are two groups,  $g_0$  with members at  $A$ ,  $B$ ,  $E$ , and  $F$ , and  $g_1$  with members at  $B$ ,  $E$ , and  $F$ . Two trees,  $t_0$  (with leaves at  $A$ ,  $B$ ,  $E$ , and  $F$ ) and  $t_1$  (with leaves at  $B$ ,  $E$ , and  $F$ ) are constructed. According to the previous definitions, we can see  $t_0$  is a perfect match for  $g_0$ , and  $t_1$  is a perfect match for  $g_1$ , while  $t_0$  is a leaky match for  $g_1$ . If only perfect match is allowed, we need both trees to cover groups  $g_0$  and  $g_1$ . On the other hand, if leaky match is allowed, we can use  $t_0$  for both groups. Clearly, by leaky match, we can achieve better aggregation (i.e., using fewer trees to cover more groups). A disadvantage of leaky match is that some bandwidth is wasted to deliver data to nodes that are not members of the group. For example, when we use  $t_0$  to cover both groups, data for group  $g_1$  will be delivered to node  $A$ , which has no members of this group. Therefore, some bandwidth is wasted on the link between nodes  $A$  and  $C$  (denoted as  $l_{AC}$ ). In this way, we trade off bandwidth for state scalability.

Theoretically, the group-to-tree matching problem is an intractable multi-objective (minimizing bandwidth waste and maximizing aggregation) optimization problem, with the two objectives contradicting each other. In reality, however, a network manager could seek efficient group-to-tree matching algorithms which achieves the best aggregation while keeping bandwidth waste under some given threshold. Alternatively, we can minimize bandwidth waste while maintaining a predefined number of trees (i.e., fixing the aggregation level). In this paper, we investigate the former version of the problem, and we

plan to tackle the latter version in our future work.

## 2.2 Network Model and Definitions

### 2.2.1 Network Model

The network is modelled as an undirected graph  $N(V, E)$ . 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  $t$ , the total cost to distribute a unit amount of data over tree  $t$  is

$$C(t) = \sum c_{ij}, \text{ link } (i, j) \in t. \quad (1)$$

For the simplicity of presentation, we assume every link has equal cost 1, then the tree cost is simply  $C(t) = |t| - 1$ , where  $|t|$  denotes the number of nodes in  $t$ <sup>1</sup>.

### 2.2.2 Bandwidth Waste

Now consider a network  $N(V, E)$ , in which a multicast routing algorithm  $A$  (e.g., the shortest path tree algorithm) is used to set up multicast trees. Given a multicast group  $g$ , we can compute a multicast tree  $t^n(g)$  (referred to as the “native” tree for group  $g$ ) using the routing algorithm<sup>2</sup>. If this group is covered by an aggregated tree  $t(g)$ , the **bandwidth waste** is defined as

$$\delta(t, g) = \frac{C(t(g)) - C(t^n(g))}{C(t^n(g))}. \quad (2)$$

This metric reflects the relative additional bandwidth when tree  $t(g)$  is used to carry data for group  $g$ , compared with using the native tree  $t^n(g)$ . The implication of this metric is that, for a given group, there are multiple trees to cover all the members of this group. If tree aggregation is not taken into

<sup>1</sup> It is important to note that this assumption does not invalidate our algorithms for networks where links have non-equal costs. In the group-tree matching algorithms, the key component related to link costs is the computation of bandwidth waste, which is explained clearly for both the cases of equal cost and non-equal cost in Section 2.2.2.

<sup>2</sup> Here we assume each group has only one native tree. When a group has more than one native tree due to multiple sources (or cores), we can convert a group with different sources (or cores) into a set of groups each of which has a unique source (or core) and a native tree and apply the proposed group-to-tree matching algorithms on the generated groups.

consideration, trees with lower cost would be preferred, because data delivery on these trees incurs lower bandwidth cost. However, in order to aggregate more groups into a tree, we may need to use large trees that can cover other groups for this group. This metric is used to quantify the cost we use large trees for a group. For instance, in Fig. 1, assuming  $t_1$  is the native tree for group  $g_1$ , the bandwidth waste of  $t_1$  is 0 for  $g_1$ . On the other hand, if  $t_0$  is used to cover  $g_1$ , some bandwidth is wasted on the link  $l_{AC}$ , because there are no members at node A. In this case, the bandwidth waste can be computed in the following way:

$$\delta(t_0, g_1) = \frac{C(t_0) - C(t_1)}{C(t_1)} = \frac{C(l_{AC})}{C(l_{CE}) + C(l_{CD}) + C(l_{BD}) + C(l_{DF})}. \quad (3)$$

Following the assumption of equal link cost of 1, bandwidth waste can be represented as

$$\delta(t, g) = \frac{|t(g)| - |t^n(g)|}{|t^n(g)| - 1}. \quad (4)$$

For the above example, the bandwidth waste of  $t_0$  for  $g_1$  is  $\delta(t_0, g_1) = 1/4$ .

To control the amount of bandwidth waste, in our group-to-tree matching problem, a group  $g$  is allowed to be mapped onto a tree  $t$  only if  $\delta(t, g) \leq bth$ , where  $bth$  is a pre-determined bandwidth waste threshold (per match). Note that,  $t^n(g)$  is not necessarily the minimum cost tree (Steiner tree). In fact, many practical multicast routing algorithms use shortest path tree (e.g., MOSPF [22], DVMRP/PIM-DM [15], PIM-SSM [19]) or core-based tree (e.g., CBT [4] and PIM-SM [15]). Therefore, the aggregated tree  $t(g)$  may happen to be more efficient than  $t^n(g)$ . Thus, it is possible for  $\delta(t, g)$  to be negative. It is worth mentioning that, our goal in this paper is not to propose a multicast routing algorithm to compute a multicast tree for a given group; instead, we assume we are given a multicast routing algorithm, and we want to compute a minimum number of trees for a given set of groups without a significant amount of bandwidth waste.

### 2.2.3 Aggregation Degree

Let  $N_g$  be the number of multicast groups in the network and  $N_t$  the number of aggregated trees used to cover those groups, **Aggregation Degree** is defined as

$$AD = \frac{N_g}{N_t}. \quad (5)$$

The bigger  $AD$  is, the fewer aggregated trees we need to manage, and thus the less resource usage and control overhead. In fact, maximizing aggregation degree under a given bandwidth waste threshold is the optimization goal of our group-to-tree matching algorithms.

### 2.3 Problem Formulation

We formulate the group-to-tree matching problem into two versions: static pre-defined tree version, if Aggregated Multicast is used by an ISP for tree pre-dimensioning based on long-term traffic measurement; and dynamic on-line tree version, if Aggregated Multicast is employed in on-line systems where groups dynamically join and leave.

#### 2.3.1 Static Pre-Defined Trees

In this version of the problem, we are given: a network  $N(V, E)$ , a set of multicast groups  $G$ , a multicast routing algorithm  $A$ , and a bandwidth waste threshold  $bth$ . The goal is to find  $N$  trees (each covering a different set of nodes) and a matching from a group  $g$  to a tree  $t(g)$  such that every group  $g$  is covered by  $t(g)$ , with the objective of minimizing  $N$  (equivalent to maximizing aggregation degree) while keeping the bandwidth waste under the given threshold  $bth$ . This is the problem we need to solve in order to build a set of pre-defined aggregated trees based on long-term traffic measurement information.

In fact, we can show the static pre-defined tree problem is NP-complete. Before we give the proof, we first introduce a well-studied NP-complete problem: MINIMUM SET COVER [10].

- INSTANCE: A collection  $C$  of subsets of a finite set  $S$ .
- SOLUTION: A set cover for  $S$ , i.e., a subset  $C' \subseteq C$  such that every element in  $S$  belongs to at least one member of  $C'$ .
- MEASURE: Cardinality of the set cover, i.e.,  $|C'|$ .

**Theorem 1.** The Static Pre-Defined Tree problem is NP-Complete.

**Proof.** It is easy to show that the Static Pre-Defined Tree problem is NP. Restating this optimization problem as a decision problem, we want to determine if there are  $k$  trees which can cover all the groups  $G$  within the bandwidth waste threshold  $bth$  for a given size  $k$ . Suppose we are given  $k$  trees, to validate if these trees can cover the groups  $G$  without violating  $bth$  can be performed in polynomial time.

We prove that the Static Pre-Defined Tree problem is NP-hard by showing that a special case (with reduced input space) of this problem is actually a MINIMUM SET COVER problem. We specify the special case as follows: target trees can only be selected from a given set of trees instead of all possible trees in the network. We denote the given set of trees as  $T$ . For each tree  $t_i \in T$ , we represent the group set covered by  $t_i$  with bandwidth waste below  $bth$  as  $G^c(t_i)$ , where  $1 \leq i \leq |T|$ . We denote the collection of  $G^c(t_i)$  as  $G^c(T)$ . Now, starting from an instance of the MINIMUM SET COVER problem with set  $S$  and set  $C = \{C_1, C_2, \dots, C_M\}$ , where  $C_j \subseteq S$  for  $1 \leq j \leq M$ , we can formulate an instance of the reduced-input-space case of the Static Pre-Defined Tree problem:

- Let  $S$  be the multicast group set  $G$ .
- Let  $M$  be the number of given trees  $|T|$ .
- Let  $C_i$  be the group set covered by a tree  $t_i$  (with the given bandwidth waste threshold), i.e., we treat  $C$  as the collection  $G^c(T)$ .

Then the objective of finding the minimum number of trees to cover the groups  $G$  is equivalent to minimizing the cardinality of set  $C'$ , where  $C' \subseteq C$  such that every element in  $S$  belongs to at least one member of  $C'$ . In other words, the discussed special case of the Static Pre-Defined Tree problem is NP-hard. The Static Pre-Defined Tree problem is obviously no easier than this special case, in which the input space is reduced (only a given set of trees are examined). Thus, the Static Pre-Defined Tree problem is NP-hard. This completes the proof.

To tackle this NP-complete problem, we propose three algorithms, one optimal algorithm using ILP, one near-optimal algorithm using greedy method, and one pseudo-dynamic algorithm. We will present these algorithms in Section 3 and 5 respectively.

### 2.3.2 Dynamic On-Line Trees

The dynamic version of the group-to-tree matching problem is useful for on-line systems. The goal is to find an algorithm to generate and maintain (e.g., establish, modify and tear down) a set of trees and map a group to a tree when the group joins the network, with the same objective of minimizing the number of aggregated trees without violating the given bandwidth waste threshold  $bth$ . Some previous study [26,14,13] has proposed heuristic dynamic algorithms, but there is neither comparison with optimal solutions nor comparison between themselves. In Section 4, we will first describe a generic dynamic on-line algorithm, and then compare its performance with optimal solutions and other heuristics.

Table 1  
 Definitions of variables in algorithms

Variable	Definition
$bth$	bandwidth waste threshold
$\delta(t, g)$	bandwidth waste between tree $t$ and group $g$
$l$	the number of links that can be extended without violating $bth$
$t^n(g)$	native tree of group $g$
$t^f(g)$	final tree selected for group $g$
$t^e$	extended tree
$T^c(g)$	candidate trees for group $g$
$T^c$	candidate trees for all the groups
$G^c(t)$	groups that can be covered by tree $t$ without exceeding $bth$
$T^s$	trees selected from candidate trees to cover groups
$T$	existing trees used to cover groups that have already joined

### 3 Algorithms for Static Pre-Defined Trees

In this section, we present two algorithms for the Static Pre-Defined Tree problem. The basic idea behind these algorithms is to first find the candidate trees, with each covering a subset of the given multicast groups, then convert the problem of selecting the minimum number of trees for all the groups to a classical MINIMUM SET COVER problem, and finally map each group to a selected tree. Thus, we divide the problem into three sub-problems, namely, candidate tree generation, tree selection and group-to-tree mapping. The candidate tree generation and group-to-tree mapping sub-problems can be solved in polynomial time, while the tree selection sub-problem is in fact NP-complete. We present an integer linear programming (ILP) based optimal algorithm and a greedy algorithm for the tree selection sub-problem. For simplicity, we call the algorithm using ILP approach for tree selection as **ILP** algorithm, and the one using greedy approach for tree selection as **Greedy** algorithm.

In the following subsections, we describe the algorithms to solve each of the sub-problems, and then analyze the time complexity of these algorithms. We list the definitions of the variables used in our algorithms in Table 1.

### 3.1 Candidate Tree Generation

The sub-problem of candidate tree generation can be formulated as follows: given a network graph  $N(V, E)$ , a multicast routing algorithm  $A$  used to set up multicast trees<sup>3</sup>, and a set of groups  $G$ , for each multicast group  $g \in G$ , find the set of trees  $T(g)$  that can cover  $g$  while satisfying the bandwidth waste threshold  $bth$ , since this set of trees are the potential candidate trees that  $g$  can be mapped to.

Our algorithm works as follows. For each group  $g \in G$ , its native tree  $t^n(g)$  (using multicast algorithm  $A$ ) is first computed. This native tree is certainly one of the potential trees for group  $g$ , so it belongs to the candidate tree set  $T^c(g)$ . Then this native tree is extended to generate more potential trees with no more than  $l$  additional links<sup>4</sup>, where  $l$  is determined by the number of links on the native tree (denoted as  $|t^n(g)|-1$ ) and the bandwidth waste threshold  $bth$ :  $l = (|t^n(g)| - 1) \times bth$ . To extend a tree  $t$ , the algorithm recursively checks for each non-tree node  $v$  whether the newly extended shortest path tree including  $v$  is a candidate tree (i.e., the number of additional links compared with  $t^n(g)$  is no more than  $l$ ): if it is, then the tree can be extended to contain  $v$ . The generated trees are also included in  $T^c(g)$ . After the candidate trees for all groups have been generated, for each candidate tree  $t$ , the set of groups  $G^c(t)$  that can be covered by  $t$  is computed.

### 3.2 Tree Selection

Based on the mapping between candidate trees and groups, we need to select a minimum set of trees such that every group can be mapped to at least one tree in this set. By dividing the set of multicast groups  $G$  into a number of subsets  $G^c(t)$ , each of which contains the groups that can be covered by the same candidate tree  $t$ , the tree selection problem now becomes how to find the minimum number of subsets that cover the original  $G$ . In this way, we reduce

---

<sup>3</sup> In this paper, we use shortest path tree algorithm as an example since many practical multicast routing algorithms use shortest path tree (e.g, MOSPF, DVMRP/PIM-DM, PIM-SSM) or core-based tree (e.g., CBT and PIM-SM), which is essentially a shortest path tree algorithm rooted at the core. For simplicity, we assume there is a unique shortest path between every two nodes (a metric can be used as tie-breaker when there are more than one shortest paths); thus, the shortest path tree algorithm returns a unique multicast tree for a group.

<sup>4</sup> Since we only consider shortest path trees, a bigger shortest path tree which covers  $g$  must be an extended tree containing  $t^n(g)$ . Note that, for other multicast routing algorithm, such as minimum spanning tree, candidate trees can not be obtained by tree extension.

this problem to the MINIMUM SET COVER problem. On the other hand, similar to Theorem 1, we can easily prove that the MINIMUM SET COVER problem can be reduced to the tree selection problem, i.e., the tree selection problem is NP-Complete. In the following, we present an ILP approach and a greedy approach for the tree selection problem.

### 3.2.1 ILP Approach

An advantage of ILP is that the problems expressed in ILP formulation with limited variables and constraints can be solved optimally with free or commercially available softwares. Therefore, our major task is to develop ILP formulation for the tree selection problem. Before presenting the ILP formulation, we define the following variables:

$$x_j = \begin{cases} 1, & \text{if tree } t_j \text{ is selected} \\ 0, & \text{otherwise} \end{cases} \quad \forall j \in [1, N_t]$$

$$c_{ij} = \begin{cases} 1, & \text{if } g_i \text{ is covered by } t_j \\ 0, & \text{otherwise} \end{cases} \quad \forall i \in [1, N_g], j \in [1, N_t]$$

where  $N_g$  and  $N_t$  are the total number of groups and candidate trees, respectively.

The objective is to minimize the number of selected trees:

$$\min \sum_{j=1}^{N_t} x_j$$

subject to the constraint that every group must be covered by at least one tree:

$$\sum_{j=1}^{N_t} c_{ij} \geq 1 \quad \forall i \in [1, N_g]$$

By solving the ILP, we obtain the values for  $x_j$  ( $j \in [1, N_t]$ ), from which we can determine the set of trees  $T^s$  to be selected.

### 3.2.2 Greedy Approach

Although the ILP approach is able to find out optimal solutions for small problems, its computation overhead makes it infeasible for relatively large

problems. Thus, we also present a greedy algorithm which reduces computation time significantly with slight performance degradation. As shown in Algorithm 1, given the multicast groups  $G$ , the candidate trees  $T^c = \cup_{g \in G} T^c(g)$ , and the mapping between each tree  $t$  and its covered groups  $G^c(t)$ , this algorithm iteratively finds a tree that covers the largest number of uncovered groups, adds it into the selected tree set  $T^s$ , and removes these groups from  $G$ . This algorithm terminates when  $G$  is empty, which means all the groups have been covered by at least one tree.

---

**Algorithm 1** Greedy( $G, T^c, G^c$ )

---

```

1:  $T^s \leftarrow \emptyset$  // initialize selected trees
2: while  $G$  is not empty do
3:    $best\_tree \leftarrow 0$  // currently best tree
4:    $max\_cover \leftarrow 0$  // num. of groups covered by  $best\_tree$ 
5:   for all  $t \in T^c$  do
6:     if  $|G^c(t) \cap G| > max\_cover$  then
7:       //  $t$  covers more uncovered groups than  $best\_tree$ 
8:        $max\_cover \leftarrow |G^c(t)|$ 
9:        $best\_tree \leftarrow t$ 
10:    end if
11:  end for
12:   $T^s \leftarrow T^s \cup \{best\_tree\}$ 
13:   $T^c \leftarrow T^c - \{best\_tree\}$ 
14:   $G \leftarrow G - G^c(best\_tree)$ 
15: end while
16: return  $T^s$ 

```

---

### 3.3 Group-To-Tree Mapping

Given the groups  $G$  and the selected trees  $T^s$ , a group  $g$  may be covered by multiple trees in  $T^s$ . In this case, we break the tie by selecting the tree that can cover  $g$  with the minimum bandwidth waste as the final tree  $t^f(g)$  for this group.

### 3.4 Complexity Analysis

Since the tree selection problem is NP-complete, we here focus on the complexity analysis of the Greedy algorithm. To facilitate our analysis, we define several variables as shown in Table 2. We assume that the shortest paths between every pair of nodes are already obtained from the underlying unicast routing algorithm, and we do not consider the time complexity of native multicast tree computation since it is determined by the given multicast routing

Table 2

Definition of variables in complexity analysis

Variable	Definition
$m$	number of nodes
$n$	number of links
$N_g$	number of groups
$N_{tc}$	number of candidate trees
$N_{ts}$	number of selected trees ( $N_{ts} \leq N_g$ )
$N_t$	number of existing trees ( $N_t \leq N_g$ )

algorithm.

In the candidate tree generation algorithm, given a tree  $t$ , finding all the trees that are extended from  $t$  with one more node requires scanning every non-tree node  $v$  and deciding if the extended tree satisfies bandwidth waste threshold requirement, which takes  $O(\binom{m}{1}m) = O(m^2)$  time. Since adding one node introduces at least one more link, we can derive that the operation of finding trees extended from  $t$  with additional  $l$  links involving at most  $l$  times of node scanning, which takes  $O(\binom{m}{l}m) = O(\frac{m!}{(m-l)!}m)$ . Therefore, the time complexity of finding the extended trees with no more than  $l$  links is:

$$O\left(\sum_{i=1}^l \frac{m!}{(m-i)!i!}m\right) \ll O\left(\sum_{i=0}^m \frac{m!}{(m-i)!i!}m\right) = O(m2^m)$$

Thus, it seems that this algorithm in general is not polynomial time. However, in practice, the value of  $l$  is restricted to be relatively small in order to control bandwidth waste. If we assume  $l \leq 3$ , the time complexity of generating candidate trees for one group becomes:

$$O\left(m^2 + \frac{m(m-1)}{2}m + \frac{m(m-1)(m-2)}{6}m\right) = O(m^4)$$

In this way, the candidate tree generation algorithm takes  $O(N_g m^4)$  time to find the candidate trees for all the groups. Using the same reasoning, we can conclude that, when  $l \leq 3$ , the total number of candidate trees  $N_{tc}$  is bounded:  $N_{tc} = O(N_g m^3)$ .

For the tree selection sub-problem, the Greedy algorithm requires scanning all the candidate trees for  $N_{ts}$  times until all  $N_{ts}$  trees are selected. Obviously,  $N_{ts} \leq N_g$ , so the time complexity of this algorithm is  $O(N_{tc} N_{ts}) = O((N_g m^3) N_g) = O(N_g^2 m^3)$ .

Finally, in the group-to-tree mapping algorithm, for each group, the selected trees  $T^s$  are searched to find a best tree. Since it takes at most  $O(n)$  time to check if a tree covers a group and compute the bandwidth waste between a group and a tree, this algorithm needs  $O(N_g N_{ts} n) = O(N_g^2 n)$  time.

Combining the three sub-problems, we conclude that, when the bandwidth waste threshold  $bth$  is reasonably small, the time complexity of the Greedy algorithm is polynomial.

**What to do when  $bth$  is big?** Obviously, when the bandwidth waste threshold  $bth$  is big, the Greedy algorithm will become time consuming. Inspired by the dynamic on-line group-to-tree matching heuristics, we propose a pseudo-dynamic algorithm which will be presented in Section 5.

## 4 A Generic Dynamic On-Line Algorithm

The algorithms discussed above are expected to achieve optimal or near-optimal performance in minimizing the number of aggregated trees with the given bandwidth waste constraint. However, they are used to match a static set of groups onto trees, and thus cannot be used as “on-line” algorithms. A dynamic on-line group-to-tree matching algorithm needs to accommodate group dynamics while satisfying the same goal. At the same time, the complexity of the algorithm is also a major concern, since on-line systems usually require fast response to user requests. In this section, we describe a generic dynamic on-line algorithm (referred to as **Dynamic** algorithm) which extends the various heuristics in the literature, and provide an upper-bound analysis on its performance.

### 4.1 Algorithm Description

Considering group dynamics, we present the algorithm in two procedures: group join and group leave. As shown in Algorithm 2, when a new group  $g$  joins the network, we first identify if there are existing trees to cover it. For each existing tree  $t \in T$ , if it can cover  $g$  without exceeding  $bth$ , then  $t$  is a candidate tree. If  $t$  cannot cover  $g$ , we compute an extended tree  $t^e$  based on  $t$  to cover the nodes in  $g$  that are not covered by  $t$ . Since tree extension increases the bandwidth waste between  $t^e$  and the groups mapped to the original tree  $t$  (denoted as  $G^c(t)$ ), we must check whether tree  $t^e$  satisfies the bandwidth waste requirement for all of the groups mapped to  $t$ , i.e.,  $G^c(t) \cup \{g\}$ . If this is the case, then  $t^e$  is added to the candidate tree set (but the existing tree set  $T$  and the original tree  $t$  does not change). Finally, If there is no candidate tree,

the native tree  $t^n(g)$  of this group is constructed as its final tree. Otherwise, the candidate tree with the minimum bandwidth waste is selected as the final tree  $t^f(g)$ . If this final tree  $t^f(g)$  is extended from an existing tree  $t'$ , we update the existing tree set  $T$  by removing the original tree  $t'$  and adding the extended tree  $t^f(g)$ , and use  $t^f(g)$  to cover the groups originally covered by  $t'$ . An example is shown in Fig. 2 to illustrate the join algorithm ( $bth = 0.4$  in this example). In Fig. 2(a), there is initially one group  $g_0$ . Because there are no existing trees, the native tree  $t_0$  is established for  $g_0$ . In Fig. 2, when group  $g_1$  arrives,  $t_0$  can be extended to  $t_1$  to cover  $g_1$ . Because the bandwidth waste of  $t_1$  for  $g_0$  and  $g_1$  is  $1/3$ , which is smaller than the threshold  $bth$ ,  $t_1$  is used for both  $g_0$  and  $g_1$ .

---

**Algorithm 2** Dynamic-Join( $g$ )

---

```

1:  $T^c(g) \leftarrow null$  // initialize candidate tree set
2: compute a native tree  $t^n(g)$  for  $g$ 
3: for all  $t \in T$  do
4:   //  $T$  is the existing tree set
5:   if  $t$  covers  $g$  then
6:     if  $\delta(t, g) \leq bth$  then
7:        $T^c(g) \leftarrow T^c(g) \cup \{t\}$ 
8:     end if
9:   else
10:    compute extended tree  $t^e$  from  $t$  to cover group  $g$ 
11:    if  $\delta(t^e, g) \leq bth$  for  $g' \in G^c(t) \cup \{g\}$  then
12:      //  $G^c(t)$  is the set of groups assigned to  $t$ 
13:       $T^c(g) \leftarrow T^c(g) \cup \{t^e\}$  //  $t$  and  $T$  remain the same
14:    end if
15:  end if
16: end for
17: if  $T^c(g) == null$  then
18:    $t^f(g) \leftarrow t^n(g)$ 
19: else
20:    $t^f(g) \leftarrow t \in T^c(g)$  with min.  $\delta(t, g)$ 
21:   if  $t^f(g)$  is extended from tree  $t' \in T$  then
22:     $T \leftarrow T \cup \{t^f(g)\} - \{t'\}$ 
23:   end if
24: end if
25: return  $t^f(g)$ 

```

---

When a group leaves the network, Algorithm 3 will be activated. When group  $g$  leaves, the bandwidth waste of other groups will not increase, i.e., the bandwidth waste constraint will not be violated. However, in order to obtain a better set of trees to cover the active groups, the tree  $t(g)$  will be shrunk. The shrinking option is described in line 5. For example, in Fig. 2(c), when group  $g_0$  leaves,  $t_1$  can be shrunk to  $t_2$  while still covering  $g_1$ . The advantage

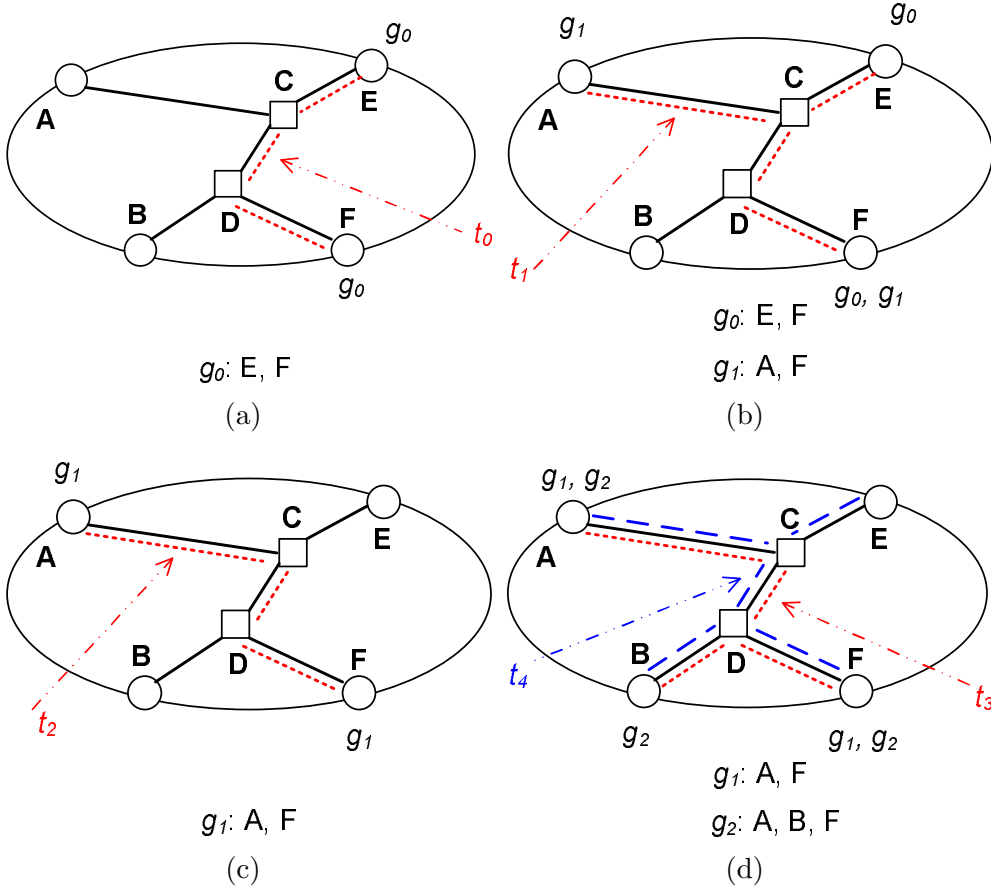


Fig. 2. An illustration of the Dynamic algorithm ( $bth = 0.4$ ). (a) When group  $g_0$  joins,  $t_0$  is established. (b) When group  $g_1$  joins,  $t_0$  is extended to  $t_1$ . (c) When group  $g_0$  leaves,  $t_1$  is shrunk to  $t_2$ . (d) When group  $g_2$  joins,  $t_2$  is extended to  $t_3$ .

of shrinking tree can be illustrated by Fig. 2(d). In this case, group  $g_2$  arrives, and we can extend the tree  $t_2$  to  $t_3$  to accommodate both  $g_1$  and  $g_2$ . The bandwidth waste of  $t_3$  for these two groups is lower than the bandwidth waste threshold. However, if  $t_1$  had not been shrunk when  $g_0$  leaves, in order to cover the members of  $g_2$ ,  $t_1$  would have been extended to  $t_4$ , which nevertheless violates the bandwidth waste constraint for group  $g_1$  (the bandwidth waste is  $2/3$ ). Therefore, if we do not shrink tree  $t_2$  in Fig. 2(c), two trees will be needed for  $g_1$  and  $g_2$ .

---

**Algorithm 3** Dynamic-Leave( $g$ )

---

- 1: identify the tree  $t(g)$  which covers  $g$
  - 2: **for all**  $g' \in G^c(t(g))$  **do**
  - 3: mark all the nodes used to deliver data for  $g'$
  - 4: **end for**
  - 5: delete all unmarked nodes from  $t(g)$  //shrink  $t(g)$
- 

It is clear that the Dynamic algorithm is not an optimal solution, because it heuristically extends or shrinks trees whenever a group joins or leaves the

network. Nevertheless, it is more practical for on-line systems when groups dynamically join and leave the network. In the following subsections, we evaluate the complexity and performance

#### 4.2 Complexity Analysis

In the above Dynamic algorithm, the existing trees  $T$  are searched to find a best tree for each group. Checking the eligibility and bandwidth waste of an existing tree or an extended tree, extending a tree, and finding the native tree can all be achieved in  $O(n)$  time. So, if we denote the number of existing trees as  $N_t$  (i.e.,  $|T|$ ), the time complexity for group join is  $O(N_t n) = O(N_g n)$  ( $N_t$  is clearly bounded by  $N_g$ ). A similar analysis applies to group leave, which has the same time complexity of  $O(N_g n)$ .

#### 4.3 An Upper Bound

To design a good dynamic on-line algorithm, one important concern is to decide how good the algorithm is, or how far this algorithm degrades from the optimal solution. However, for the dynamic group-to-tree matching problem, it is very difficult to have a global optimal solution due to group dynamics. Therefore, we obtain an upper bound on the performance of the dynamic on-line algorithms by acquiring an optimal solution at any given time point: at every sampling point, we collect the set of currently active groups, run an optimal (or near-optimal) Static Pre-Defined Tree algorithm, and decide the minimum set of trees to cover these groups. This set of trees is guaranteed to provide an upper bound on the tree set obtained by the Dynamic algorithm in terms of Aggregation Degree, since Dynamic only considers the existing trees as candidates, while the optimal solution takes all possible trees into account. In Section 6, we will investigate the performance of the Dynamic algorithm by comparing it with the Greedy algorithm for Static Pre-Defined Trees, as the Greedy algorithm provides a near-optimal solution with a much faster computation speed compared with the optimal ILP algorithm.

#### 4.4 Discussions on Existing Dynamic Heuristics

In the literature, several dynamic on-line group-to-tree matching heuristics have been proposed [26,14,13]. In [26], Song et. al. developed a protocol called **MTBF** (Multicast Tunnelling with Branching Forwarding) for the inter-domain multicast architecture MASC/BGMP. The basic algorithm in the MTBF scheme can be abstracted as follows: for a domain with  $n_e$  edge

routers, only  $n_e$  aggregated trees will be established, each rooted at one edge router. Groups with the same source edge router will share one tree. In other words, this protocol fixes the number of aggregated trees as  $n_e$ , no matter how many groups exist in the domain. To reduce bandwidth waste caused by the “large” trees, filters are added in routers to stop unnecessary data forwarding for some groups. However, adding filters is equivalent to increasing group state, i.e., the state reduction will be sacrificed. Besides the MTBF protocol, in [14,13], ASSM and BEAM are presented for single-source and multiple-source applications respectively in backbone domain multicast provisioning. They essentially use a simplified version of the Dynamic algorithm without tree extension and shrinking in order to reduce overhead. In this paper, we refer to this simple version of Dynamic as **Simple-Dynamic**. In Section 6, we will quantitatively compare the performance of the Dynamic algorithm with the MTBF and Simple-Dynamic algorithms.

#### 4.5 *Discussions on Member Dynamics*

For a practical on-line group-to-tree matching algorithm, it has to consider member dynamics, i.e., the membership of a group may dynamically change during the group life time. In our presented Dynamic algorithm, we assume all the members arrive and leave at the same time, only considering group dynamics. Then some natural questions are: how could the Dynamic algorithm handle member dynamics and how will these dynamics affect its performance?

In fact, we can easily extend the algorithm to handle the case when members do not join and leave simultaneously. When a member first joins a group, it activates the same procedure as the group join. If a member joins an existing group, then the algorithm needs to check if the tree covering the group can also cover this new member. In addition, it also needs to verify if the bandwidth waste threshold is still satisfied. If all these conditions hold, no further action is necessary, and the “old” tree is simply used for the data delivery for the “bigger” group. Otherwise, extended trees are first examined. If no extended trees satisfy the “bigger” group, then all the old members of the group have to leave the “old” tree, and together with the member activates a new group join procedure. As for member leaving, a similar extension applies.

Clearly, member dynamics cause additional overhead to the Dynamic algorithm, and this is also true for other group-to-tree matching dynamic heuristics (e.g., the MTBF and Simple-Dynamic schemes). In Section 6, we conduct simulations to study how member dynamics affect the dynamic algorithms.

## 5 Pseudo-Dynamic Algorithm for Static Pre-Defined Trees

In Section 3, we presented two algorithms, the ILP algorithm and the Greedy algorithm, for Static Pre-Defined Trees. Though the latter one is practical when the bandwidth waste threshold  $bth$  is small, we remark that the efficiency of the Greedy algorithm will be significantly degraded as  $bth$  is increased, due to the candidate tree generation procedure. Inspired by the dynamic on-line algorithm presented in last section, we propose a so-called **Pseudo-Dynamic** algorithm for the Static Pre-Defined Tree problem.

The basic idea of Pseudo-Dynamic algorithm is to use the Dynamic algorithm for a static set of groups. Given a static set of groups  $G$ , we randomly select a group and let it join the network. We repeat this step on the remaining groups until all groups join the network. In this way, we obtain a trace of groups dynamically joining the network, for which we can apply the Dynamic algorithm.

In the generated group trace, there is only group join and no leave. After all the groups join the network, we can obtain the proper trees covering all the groups by running the Dynamic algorithm. Obviously, the result may be far from the optimal, and can be affected by the order of group join. We can improve the performance by running the algorithm on multiple group traces. In fact, in our simulation study, we will show that this Pseudo-Dynamic algorithm is very effective: it is much faster than the Greedy algorithm when  $bth$  is big without sacrificing too much performance.

Following the previous time analysis, it is easy to deduct the time complexity for the Pseudo-Dynamic algorithm as  $O(N_g N_{ts} n) = O(N_g^2 n) = O(N_g^2 m^2)$ , which is faster than  $O(N_g^2 m^3)$  in the case of the Greedy algorithm even when the bandwidth waste threshold is small (i.e.,  $l \leq 3$ ).

## 6 Simulation Study

In this section, we first introduce our simulation environments and performance metrics, and then we present simulation results.

### 6.1 Simulation Settings

In our simulations, we use two network topologies based on two real backbone networks. The first one is abstracted from an AT&T IP backbone with 123 nodes [3]. This network consists of 9 gateway routers, 9 backbone routers, and

105 remote access routers. We conduct an abstraction procedure to generate a simplified network of 54 nodes as follows: the remote access routers attached to the same gateway or backbone router are “contracted” into one *contracted node* and one additional *exchange node* is created for each gateway router to represent the peering networks. The second one is a vBNS backbone with 43 nodes [2]. We use the abstracted AT&T topology to evaluate the static algorithms and both topologies to evaluate the dynamic algorithms.

In the obtained network topologies, we generate group instances to run different group-to-tree matching algorithms. We use *Random Node Weight Model* [17] for group member distribution. In this model, we consider the probability (which we call weight) that a router “participates” in multicast groups, that is, this router has attached end hosts that join multicast groups. We assign a weight 0 to core routers, and a weight from 0.01 to 0.9 to edge routers. For example, in the abstracted AT&T network, gateway nodes and backbone nodes are assumed to be core routers only. Access routers have weight 0.01, and the weights of contracted nodes are determined by the total weights of access routers they represent. Exchange nodes are each assigned a weight 0.9, since they usually connects to peering networks with a large number of group members. In the vBNS backbone, 16 FORE ASX-1000 nodes are treated as core routers. The weights of the remaining routers are based on the link bandwidth of the original backbone routers: we assign a weight 0.8 to nodes with OC-12C links, 0.1 to nodes with OC-3C links, and 0.01 to nodes with DS-3C links.

We assume multicast group requests arrive as a Poisson process, and the lifetime of a group follows an exponential distribution. In our simulations, we fix the average group life time as 100 seconds and the simulation time as 600 seconds, and vary the group arrival rate to control the total number of groups. The group-to-tree matching algorithms for Static Pre-Defined Trees are executed on active groups every 10 seconds, and the Dynamic algorithm is executed continuously as group joins and leaves the network.

We implement the group-to-tree matching algorithms in C++. We use *lp\_solve* (Version 5) [1], an open source (Mixed-Integer) Linear Programming system, to solve the ILP problems. An Intel Xeon 2.40GHz computer with 1GB memory is used to execute the simulation experiments.

To evaluate the performance of the group-to-tree matching algorithms, besides Aggregation Degree (AD) introduced in Section 2.2, we define the following metrics:

**State Reduction Ratio (SRR):** Since multicast state in edge routers cannot be reduced in any state reduction scheme, we only consider the state in core

routers. Thus, we define SRR as:

$$SRR = 1 - \frac{S_{agg}}{S_{no\_agg}}, \quad (6)$$

where  $S_{agg}$  and  $S_{no\_agg}$  represents the total number of multicast state entries in core routers with and without tree aggregation, respectively. The higher the SRR, the more multicast state entries are reduced.

**Program Execution Time (PET)** is the total time to run the simulation for an algorithm. It reflects the amount of computation (and sometimes memory) required for an algorithm.

**Tree Control Overhead (TCO)** is defined as the number of control messages for tree setup and removal, tree extension and shrinking in Dynamic, and filter setup and removal in the MTBF scheme [26]. It measures the control overhead of dynamic group-to-tree matching algorithms.

## 6.2 Comparing Algorithms for Static Pre-Defined Trees

We evaluate the performance of the proposed static algorithms, namely, ILP, Greedy and Pseudo-Dynamic algorithms, in the AT&T topology in the following subsections.

### 6.2.1 ILP vs. Greedy

We first compare the performance of the ILP and Greedy algorithms by varying the bandwidth waste threshold (denoted as  $bth$ ) from 0 to 0.05 and the number of concurrently active groups from 500 to 2500. Due to the computation overhead associated with the ILP algorithm, we were unable to obtain results for higher bandwidth waste thresholds and larger numbers of groups.

Figure 3 plots the results of AD (Aggregation Degree) for these two algorithms. As shown in the figure, for  $bth = 0$ , the ADs for both algorithms are exactly the same. The reason behind this is simple: under this condition, leaky match is not allowed, and each group can only be mapped onto its native tree. Thus, for both algorithms, the set of aggregated trees is the union of all the native trees. As  $bth$  increases, ILP out-performs Greedy, which is consistent with our intuition, since ILP optimally minimizes the number of aggregated trees whereas Greedy uses a heuristic algorithm to reduce the same metric. Nevertheless, we want to emphasize that the performance difference between ILP and Greedy is very small. For instance, when there are 2000 multicast groups co-existing in the network, AD is 2.601 and 2.631 for ILP and Greedy,

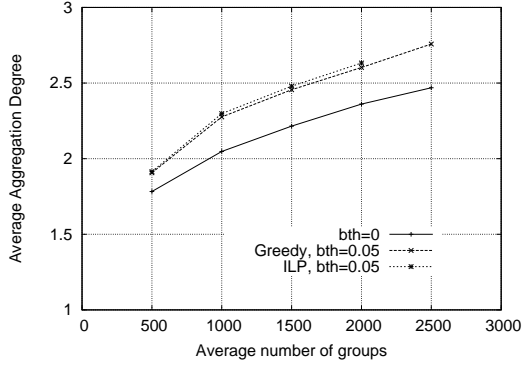


Fig. 3. AD of ILP and Greedy algorithms vs. number of concurrently active groups in the AT&T topology.

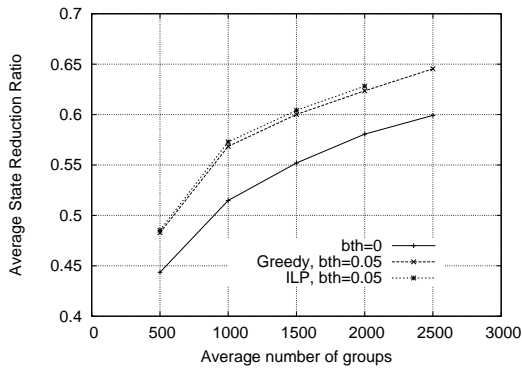


Fig. 4. SRR of ILP and Greedy algorithms vs. number of concurrently active groups in the AT&T topology.

respectively, which means ILP requires approximately 9 trees less than Greedy on average.

Figure 4 compares SRR (State Reduction Ratio) of the ILP and Greedy algorithms, and it exhibits a similar trend as Figure 3. Therefore, we can conclude that Greedy is able to achieve near-optimal performance in reducing the number of aggregation trees and multicast state entries.

Another general observation in Figures 3 and 4 is that AD and SRR increase when  $bth$  is lifted, which is exactly the trade-off Aggregated Multicast is seeking: the more bandwidth waste, the more aggregation can be achieved. Furthermore, when the number of groups changes from 500 to 2500, AD and SRR also increase, indicating Aggregated Multicast is scalable with the number of groups. When more and more groups show up in the network, the possibility of group overlapping increases, thus the aggregation degree and state reduction ratio rise. This is a general trend that can also be observed in some other figures.

Even though ILP achieves slightly higher performance than Greedy, this gain comes at a cost of additional computation overhead. To demonstrate this, we

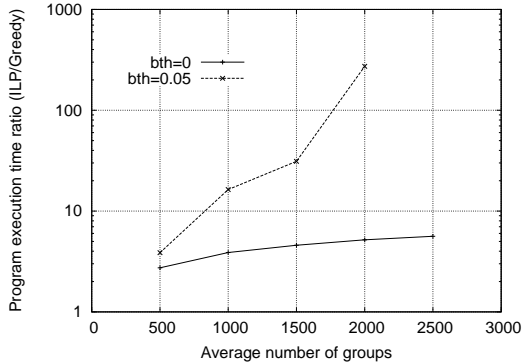


Fig. 5. PET Ratio of ILP and Greedy algorithms vs. number of concurrently active groups in the AT&T topology.

show the PET (Program Execution Time) ratio of ILP and Greedy in Figure 5. We observe that for  $bth = 0$ , the PET ratio increases approximately linearly with the number of co-existing groups, which corresponds to an exponential relationship between the time ratio and the number of groups due to the use of log-scale for the  $y$  axis; for  $bth = 0.05$ , the ratio increases even more rapidly. This indicates that the computational cost of ILP will become too expensive when there are a large number of simultaneously active groups and/or when the bandwidth waste threshold is high. For example, when  $bth = 0.05$  and the number of groups is 2500, the program is not able to finish within one day for the ILP algorithm, whereas it takes only less than 50 seconds for the Greedy algorithm to complete.

### 6.2.2 Greedy vs. Pseudo-Dynamic

As shown in the previous section, the ILP algorithm is not feasible for a large number of groups and high bandwidth waste threshold, so we now only compare the performance of the Greedy and Pseudo-Dynamic algorithms. We expect that the latter algorithm is not able to perform as well as Greedy, since it does not exploit all potential trees for a group or select trees based on a global view of the relationship between groups and trees as Greedy does. Figures 6 and 7 plot the AD and SRR for these two algorithms when the bandwidth waste threshold and the number of groups are varied. Both figures show the same trend as expected: except for  $bth = 0$  when the Greedy and Pseudo-Dynamic algorithms yield the same results for the reason explained earlier, Greedy always achieves higher multicast state scalability (or better aggregation) than the Pseudo-Dynamic algorithm. For example, when there are 4000 co-existing groups and  $bth = 0.1$ , AD is 5.400 and 4.822 for Greedy and Pseudo-Dynamic, respectively, and the corresponding SRR is 0.819 and 0.796. Additionally, as the bandwidth waste threshold increases, the difference in these two metrics becomes more distinct. We want to point out here that the performance penalty resulted from using Pseudo-Dynamic instead of Greedy is

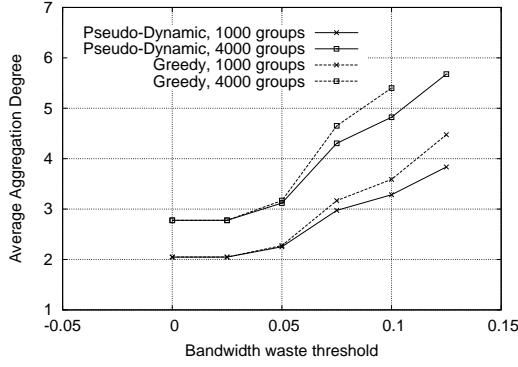


Fig. 6. AD of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold in the AT&T topology.

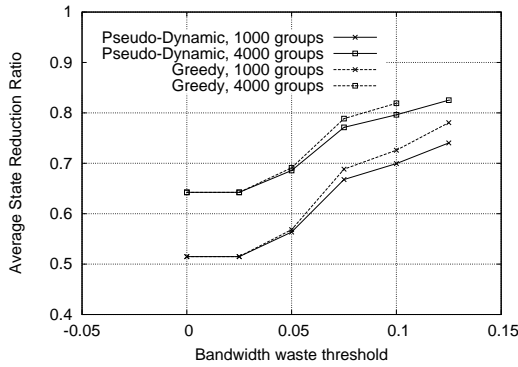


Fig. 7. SRR of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold in the AT&T topology.

very small: under the conditions tested, in the worst case, the Pseudo-Dynamic algorithm performs 14.3% worse for AD and only 5.16% worse for SRR than the Greedy algorithm.

Finally, we plot the PET ratio of the Greedy and Pseudo-Dynamic algorithms in Figure 8. We found in our simulations that the PET of the Pseudo-Dynamic algorithm is affected mostly by the number of groups and not by the value of bandwidth waste threshold (which is not shown here due to space limitation). In contrast, as shown in Figure 8, the PET ratio increases exponentially to the bandwidth waste threshold, which indicates that the PET of the Greedy algorithm grows very fast. For small  $bth$ , the Greedy algorithm runs faster; when  $bth \geq 0.075$ , the Pseudo-Dynamic algorithm out-performs the Greedy algorithm. When  $bth = 0.125$ , for 1000 groups, the PET ratio is as high as 22.8; for more than 4000 groups, the Greedy algorithm runs out of memory due to the storage requirement of a large number of potential trees for all the groups, which is another disadvantage of the ILP and Greedy algorithms. From Figure 8, another interesting observation is that when the number of groups increases, the Greedy algorithm runs faster comparatively. However, this speed-up is mitigated when the bandwidth waste threshold  $bth$  is lifted. As

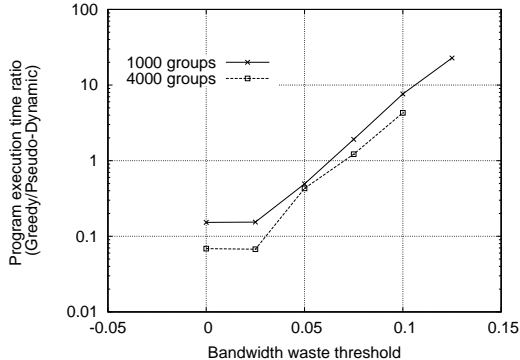


Fig. 8. PET Ratio of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold in the AT&T topology.

we can see, for 4000 groups, when  $bth \geq 0.075$ , the Pseudo-Dynamic algorithm starts to outperform the Greedy algorithm.

### 6.3 Evaluating Algorithms for Dynamic On-Line Trees

As illustrated above, among the algorithms for the static group-to-tree matching problem, Greedy yields a near-optimal solution with significantly reduced computation overhead in comparison with ILP. Therefore, we first use Greedy as an upper bound to evaluate the performance of the proposed generic Dynamic on-line algorithm in the AT&T topology. Then we compare the Dynamic algorithm with existing heuristics, namely, the Simple-Dynamic and MTBF schemes discussed in Section 4.4, in both the AT&T and vBNS topologies.

#### 6.3.1 Greedy vs. Dynamic

As mentioned earlier, the Greedy algorithm gives an upper bound of the performance of the Dynamic algorithm, because the Dynamic algorithm incrementally finds group-to-tree matching for each group as groups join and leave the network by considering only existing trees as candidates, while the Greedy algorithm considers all possible trees as candidates and tries to use one tree to cover as many groups as possible. To compare the performance of the Dynamic algorithm with the Greedy algorithm, we vary the bandwidth waste threshold  $bth$  from 0 to 0.125 and collect the simulation data of AD, SRR, and PET for different numbers of groups<sup>5</sup>.

As illustrated in Figures 9 and 10, the results match our conjecture: Greedy bounds Dynamic in all the cases. In addition, in the case of AD, as  $bth$  in-

<sup>5</sup> We tend to make  $bth$  small since the Greedy algorithm takes long time when  $bth$  is big.

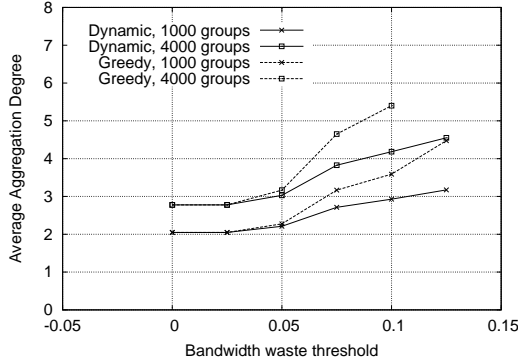


Fig. 9. AD of Greedy and Dynamic algorithms vs. bandwidth waste threshold in the AT&T topology.

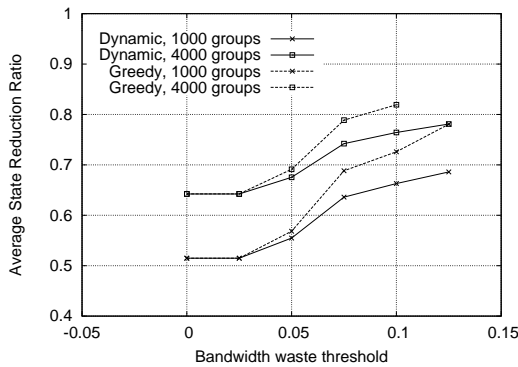


Fig. 10. SRR of Greedy and Dynamic algorithms vs. bandwidth waste threshold in the AT&T topology.

creases, the difference between these two algorithms is magnified; however, for SRR, their difference remains approximately the same for  $bth \geq 0.05$ . From these results, we can see that the Dynamic algorithm can achieve the goal reasonably well when  $bth$  is relatively small.

The execution time ratio of the Greedy and Dynamic algorithms is plotted in Figure 11. We observe that the Greedy algorithm generally takes longer time, especially for large  $bth$ , due to the exhaustive search of candidate trees required by the algorithm.

### 6.3.2 Dynamic vs. Simple-Dynamic

Simple-Dynamic is a simple version of the Dynamic algorithm without tree extension and shrinking in order to avoid the additional overhead introduced by these complex tree operations. However, are these operations completely useless? What is the trade-off between Dynamic and Simple-Dynamic? To answer these questions, we conduct another set of experiments. We vary the number of groups and collect the results of AD, SRR, and TCO for different bandwidth threshold  $bth$ .

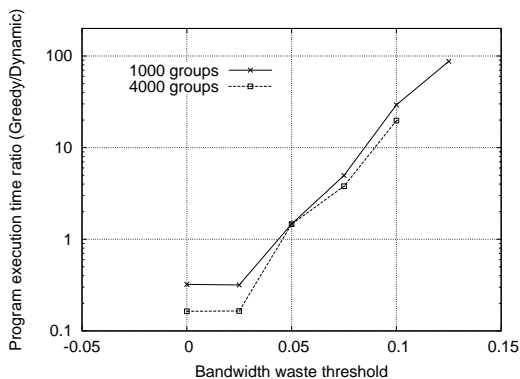


Fig. 11. PET ratio of Greedy and Dynamic algorithms vs. bandwidth waste threshold in the AT&T topology.

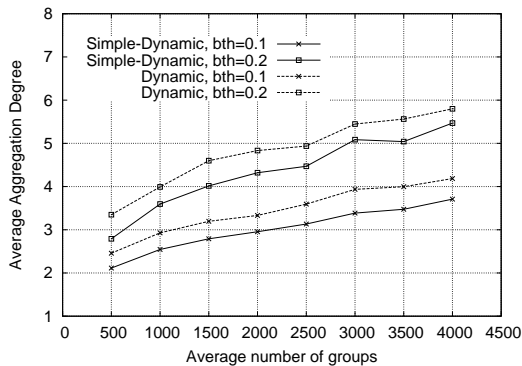
Figures 12 and 13 illustrate the results of AD (Aggregation Degree) and SRR (State Reduction Ratio) for Simple-Dynamic and Dynamic in the AT&T and vBNS topologies. These two figures show a similar trend: Dynamic achieves better performance than Simple-Dynamic. For example, in the AT&T topology, Dynamic has a SRR of 0.66 (around 10% better than Simple-Dynamic) when  $bth = 0.1$  and there are 1000 groups. Tree extension/shrinking in Dynamic tend to find a better tree set by “slightly changing” the trees, but there is no such feature in Simple-Dynamic.

In Figure 14, we see that the tree control overhead of Simple-Dynamic is higher than that of Dynamic, especially when the number of groups is big, which is counterintuitive. The explanation is that, in Dynamic, even though tree extension and shrinking cause additional overhead, the aggregation gain (i.e., less tree setup and removal overhead) compensates more than the expected loss. This shows that Dynamic is actually a winner especially when there are large numbers of groups in the network.

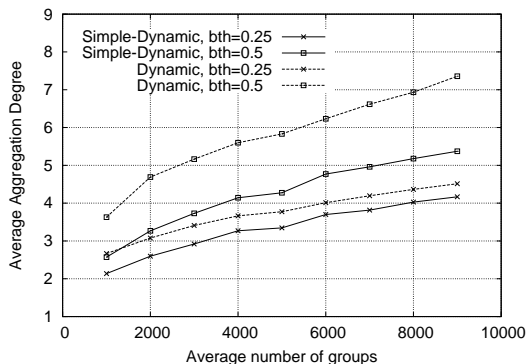
From Figures 13 and 14, we also have the following observations: when the number of groups increases, the “gain gap” is reduced, while the “cost gap” is widened. The explanation is as follows: when there are more groups in the network, usually more trees are established. Thus, it is easier to find candidate trees for a group and tree extension becomes less critical. As a result, even without extension, Simple-Dynamic tends to perform better, and gets closer to Dynamic. In terms of overhead, the “widening” gap exactly shows that Dynamic’s aggregation gain has stronger effect on control overhead than tree extension and shrinking cost.

### 6.3.3 Dynamic vs. MTBF

As we know, in the MTBF scheme, the number of trees to be maintained is fixed and filters need to be added to some routers in order to control the



(a)

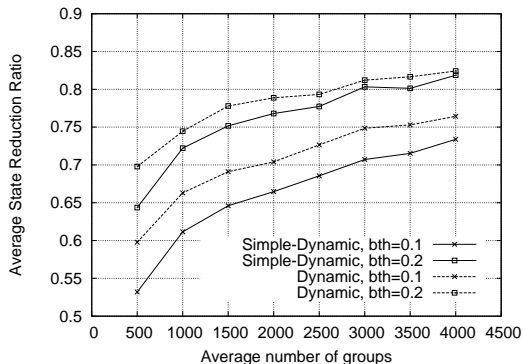


(b)

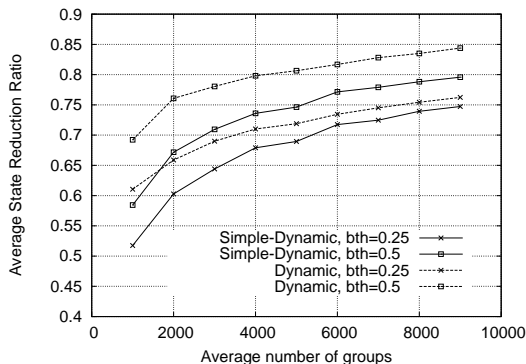
Fig. 12. AD of Dynamic and Simple-Dynamic vs. average number of groups. (a) AT&T topology; (b) vBNS topology.

bandwidth waste [26]. These installed filters are equivalent to the group state entries maintained at routers. In Figures 15 and 16, we show the results of SRR and TCO for the MTBF and Dynamic schemes when there is no bandwidth waste.

Both figures indicate that Dynamic significantly outperforms the MTBF scheme especially when there are a large number of groups in the network. We found out that the performance of the MTBF scheme depends heavily on group membership distribution: if the native tree of a group is relatively small compared with the final large multicast tree, then a large number of filters need to be installed in the multicast routers in order to avoid bandwidth waste. In other words, a very small number of aggregated trees (as in the MTBF scheme) may not be flexible enough to accommodate a large number of multicast groups without significant performance penalty. Consequently, the MTBF scheme performs poorly in our simulation experiments. Dynamic, in contrast, forces multiple groups with similar membership to share an aggregated tree, and thus is more flexible when the number of groups is large. To some extent, these results inform us that fixing the number of aggregated trees could easily sacrifice the state reduction and induce additional control overhead for



(a)



(b)

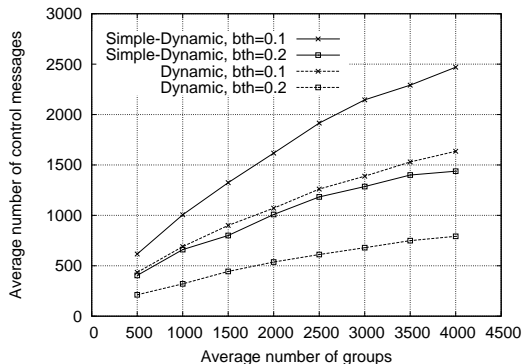
Fig. 13. SRR of Dynamic and Simple-Dynamic vs. average number of groups. (a) AT&T topology; (b) vBNS topology.

maintaining filters.

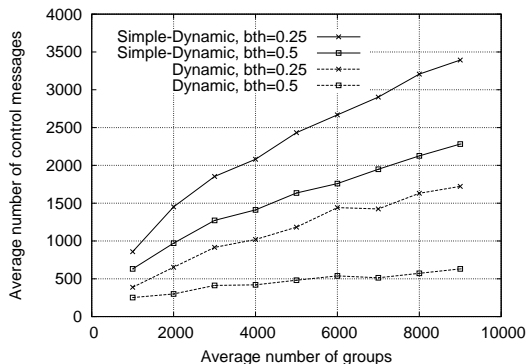
#### 6.3.4 Member Dynamics

In Section 4, we discussed the extension of the Dynamic algorithm to handle member dynamics. To evaluate the impact of member dynamics on the performance of the dynamic algorithms, we extend both the Dynamic and Simple-Dynamic algorithms and study their performance when members dynamically join and leave.

Fig. 17 and 18 plot the SRR and TCO of the Dynamic and Simple-Dynamic algorithms in the vBNS topology. Compared with Fig. 13(b) and Fig. 14(b), it is clear that member dynamics affect the performance of both algorithms: the SRR slightly decreases and the TCO dramatically increases when member dynamics are taken into consideration. These results are consistent with our intuition: when members dynamically join and leave their groups, the size of the groups will vary more significantly and the number of different combinations of group members will be higher. Therefore, these groups require a larger



(a)



(b)

Fig. 14. TCO of Dynamic and Simple-Dynamic vs. average number of groups. (a) AT&T topology; (b) vBNS topology.

number of trees to cover, which causes the SRR to decrease. Moreover, whenever the membership of a group changes, the group needs to be re-mapped onto a tree; thus, the control overhead to setup, destroy, extend and shrink trees will increase dramatically when member dynamics are considered. This conclusion also help to justify that Aggregated Multicast is mainly targeted for a transit domain, where members are relatively stable, i.e., member dynamics are much less significant than in stub domains [16].

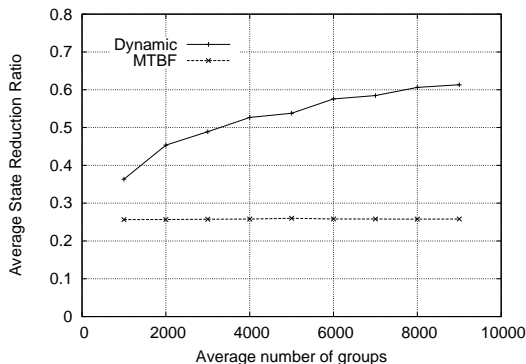
From Fig. 17 and 18, we also observe that the Dynamic algorithm performs better than the Simple-Dynamic algorithm, which is consistent with the earlier results.

#### 6.4 Summary

To summarize, our simulation study shows the following results. First, among the algorithms for Static Pre-Defined Trees, ILP achieves the best aggregation, but it is not feasible in reality due to significant processing and memory over-



(a)



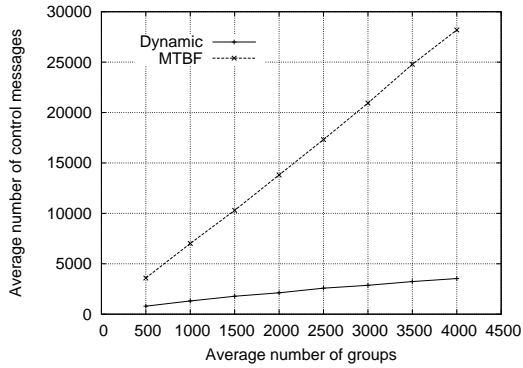
(b)

Fig. 15. SRR of Dynamic and MTBF vs. average number of groups. (a) AT&T topology; (b) vBNS topology.

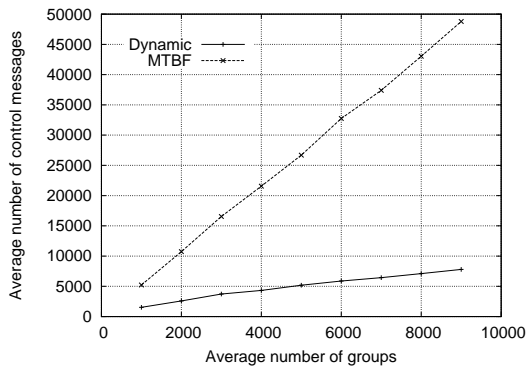
head. The performance of Greedy is very close to the optimal ILP solution. The Pseudo-Dynamic algorithm is a good heuristic where Greedy does not work well (for example, when  $bth$  is high). Second, for the dynamic group-to-tree matching problem, the generic Dynamic on-line algorithm is demonstrated to be more practical with limited performance penalty and reasonable computation requirement than some other existing heuristics. In contrast, the MTBF scheme sacrifices state reduction and introduces more control overhead when fixing the number of aggregated trees.

## 7 Related Work

State scalability problem is very challenging for multicast routing, especially in backbone networks where the number of active groups can be huge. Recently, this problem has prompted a significant number of research proposals, which can be approximately classified into three categories: (1) suppressing multicast state at some (or all) routers; (2) aggregating multicast state at individual routers; (3) Aggregated Multicast. In the following, we discuss the approaches



(a)



(b)

Fig. 16. TCO of Dynamic and MTBF vs. average number of groups. (a) AT&T topology; (b) vBNS topology.

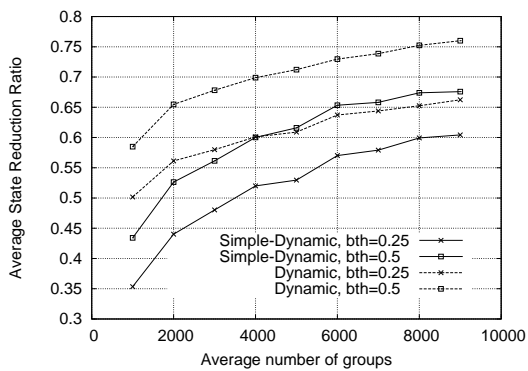


Fig. 17. SRR of Dynamic and Simple-Dynamic algorithms when group members join and leave dynamically in the vBNS topology.

in the first two categories and compare them with the aggregated multicast approach.

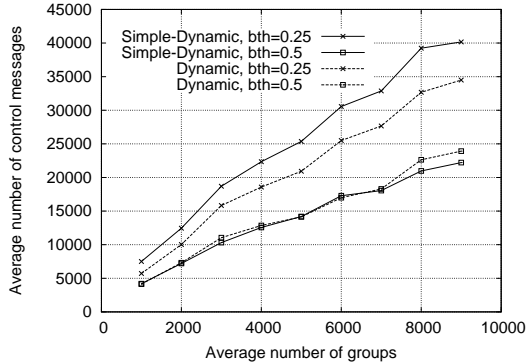


Fig. 18. TCO of Dynamic and Simple-Dynamic algorithms when group members join and leave dynamically in the vBNS topology.

### 7.1 Suppressing Multicast State at Routers

We can further divide this category into two sub-categories: (1) eliminating multicast state at non-branching routers; (2) completely removing multicast state at routers using new multicast architectures.

Examples in the first sub-category are [29], [27], and [11]. These proposals attempt to eliminate multicast forwarding state at *non-branching* routers (i.e. routers that forward multicast packets received for a group to only one outgoing interface). In [29], Tian and Neufeld propose to dynamically establish tunnels over non-branching links. In [27], Stoica et al. design a protocol called REUNITE, in which multicast state is installed at branching nodes only and packet forwarding is based on unicast in between. [11] modifies the REUNITE approach by adopting source-specific channel abstraction to simplify multicast address allocation [20]. It should be noted that all these no-branching-router dedicated schemes have a basic assumption: there are a large number of sparse groups in the network. Moreover, they only consider reducing multicast state at routers without solving the control overhead issue of the multicast state scalability problem.

Multicast architectures falling in the second sub-category aim to completely eliminate multicast state at routers using application level multicast or Xcast. In application level multicast, multicasting functionalities (such as group management and data replication) are solely implemented at end hosts and thus are transparent to routers. The proposed schemes in the literature can be classified into two broad categories: structured [25,31,7], and unstructured [9,5,23,21,18]. Structured application layer multicast schemes leverage Distributed Hash Table (DHT)-based overlay networks and build multicast forwarding trees on top of structured overlays. In unstructured schemes, self-organizing multicast trees are constructed with or without explicit underlying mesh overlays. Application level multicast is a useful alternative to IP multicast in order to facilitate

multicast wide deployment. However, it is worth noting that without explicit support from intermediate routers, multicast efficiency tends to be degraded in these schemes. It is also difficult for the unstructured approaches to support large-scale multicast applications like Internet TV that can have millions of group members, since these application level multicast protocols need to discover and maintain all or most group members.

Xcast [6] proposes coding a set of destinations' addresses in a multicast packet so that a router does not need to maintain state information for the group. However, it incurs additional processing overhead of data packets at each router and higher bandwidth overhead for transmitting larger packets. Thus, it is only suitable for very small groups.

Recently, in [30] Yang et al. combine the advantages of Xcast [6] and the non-branching node state reduction method ([29,27,11]) and design a new scheme for flexible state allocation in routers. This approach makes multicast routing more scalable to group size compared with Xcast. Additionally, balanced state distribution among routers is also considered. Similarly to all the above state suppressing approaches, however, this proposal does not solve the control overhead issue either.

## 7.2 *Aggregating Multicast State at Routers*

State reduction can also be achieved by forwarding-state aggregation, for examples, [24], [28] and [8]. Thaler and Handley analyze the aggregatability of forwarding state in [28] using an input/output filter model. Radoslavov et al. propose algorithms to aggregate forwarding state and study the bandwidth-memory tradeoff with simulations in [24]. However, this state aggregation technique attempts to aggregate routing state after the distribution trees have been established, and it tends to change the state format maintained in routers, which is generally not desired by many service providers [12]. Furthermore, the state aggregatability of this technique heavily depends on multicast address allocation.

In [8], Cheng etc. proposed a tree switching protocol to “switch” multicast trees to a pre-selected base tree. In this way, there is more overlapping among the multicast trees, thus by applying a state aggregation approach (such as [28]), more state could be reduced. However, this scheme inherits the limitation of the state aggregation method: the control overhead issue of the state scalability problem is not solved.

In short, the large number of seminal research proposals discussed above have addressed one aspect of the multicast state scalability problem: reducing mul-

unicast state at routers. In other words, they have considered the resource overhead issue. Unlike these approaches, Aggregated Multicast tends to solve both the resource and control overhead issues. In this novel method, the group-to-tree matching problem is the core problem investigated in this paper.

## 8 Conclusions

In this paper, we have studied the group-to-tree matching problem in large scale group communications. We have formulated two versions of the problem: static version and dynamic version and proved that the static version of the problem is NP-complete. For each version of the problem, we have proposed different solutions: three algorithms (ILP, Greedy, and Pseudo-Dynamic) are designed for Static Pre-Defined Trees, and one generic Dynamic on-line algorithm is presented to solve the Dynamic On-Line Tree problem. By simulation study, we find that the Greedy algorithm is a feasible solution to the Static Pre-Defined Tree problem when bandwidth waste threshold  $bth$  is small, and its performance is very close to the ILP optimal solution (less than 1.5% for most of the cases). The Pseudo-Dynamic algorithm is much more time-efficient when  $bth$  is big though it introduces some performance penalty (14.3% in the worst case of our simulations). For the Dynamic algorithm, we provide an approach to determine the upper bound on its performance and quantitatively compare its performance with some existing dynamic on-line heuristics (the Simple-Dynamic and MTBF schemes). Simulation experiments demonstrate that the Dynamic algorithm is a very practical solution to the dynamic on-line group-to-tree matching problem.

In this paper, we studied one formulation of the group-to-tree matching problem, i.e., to minimize the number of multicast trees without violating a given bandwidth waste threshold. We plan to investigate a second formulation which minimizes the bandwidth waste with a pre-determined upper bound on the total number of multicast trees. Another problem worth studying is the group-to-tree mapping subproblem. When there are multiple candidate trees for a group, we select a tree with the minimum bandwidth waste in our current algorithm. An alternative approach is to take link load into consideration. We want to study how to select trees with the goal of balancing link load in our future work.

## References

- [1] *lp\_solve, Version 5.0.*

- [2] *vBNS backbone network*. <http://www.vbns.net/>.
- [3] *AT&T IP Backbone*, 2001. <http://www.ipservices.att.com/backbone/>.
- [4] A. Ballardie. Core Based Trees (CBT version 2) multicast routing: protocol specification. *IETF RFC 2189*, Sept. 1997.
- [5] S. Banerjee, C. Kommareddy, and B. Bhattacharjee. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM*, Aug. 2002.
- [6] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, and O. Paridaens. Explicit multicast (Xcast) basic specification. *Internet draft: draft-ooms-xcast-basic-spec-01.txt*, Mar. 2001.
- [7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489 – 1499, Oct. 2002.
- [8] F. Y. Y. Cheng and R. K. C. Chang. A tree switching protocol for multicast state reduction. In *IEEE Symposium on Computers and Communications (ISCC)*, 2000.
- [9] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.
- [10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [11] L. H. M. Costa, S. Fdida, and O. C. M. Duarte. Hop-by-hop multicast routing protocol. In *Proceedings of ACM SIGCOMM*, Aug. 2001.
- [12] J. Crowcroft. Multicast Address Translation. *Internet draft: draft-crowcroft-mat-00.txt*, Nov. 2001.
- [13] J.-H. Cui, L. Lao, D. Maggiorini, and M. Gerla. BEAM: A distributed aggregated multicast protocol using bi-directional trees. In *Proceedings of IEEE ICC*, May 2003.
- [14] J.-H. Cui, D. Maggiorini, J. Kim, K. Boussetta, and M. Gerla. A protocol to improve the state scalability of source specific multicast. In *Proceedings of IEEE GLOBECOM*, Nov. 2002.
- [15] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153.
- [16] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast: an approach to reduce multicast state. In *Proceedings of Sixth Global Internet Symposium (GI)*, Nov. 2001.
- [17] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast with inter-group tree sharing. In *Proceedings of NGC*, Nov. 2001.

- [18] P. Francis. *Yoid: Extending the Multicast Internet Architecture*. White paper, <http://www.aciri.org/yoid/>.
- [19] H. Holbrook and B. Cain. Source-Specific Multicast for IP. *Internet draft: draft-holbrook-ssm-arch-03.txt*, Nov. 2001.
- [20] H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large scale single-source applications. In *Proceedings of ACM SIGCOMM*, Sept. 1999.
- [21] M. Kwon and S. Fahmy. Topology aware overlay networks for group communication. In *Proceedings of NOSSDAV*, May 2002.
- [22] J. Moy. Multicast routing extensions to OSPF. *IETF RFC 1584*, Mar. 1994.
- [23] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the Third USENIX Symposium on Internet Technologies and Systems*, Mar. 2001.
- [24] P. I. Radoslavov, D. Estrin, and R. Govindan. Exploiting the bandwidth-memory tradeoff in multicast state aggregation. Technical report, USC Dept. of CS, 99-697 (Second Revision), July 1999.
- [25] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of NGC*, Nov. 2001.
- [26] S. Song, Z.-L. Zhang, B.-Y. Choi, and D. H. Du. Protocol independent multicast group aggregation scheme for the global area multicast. In *Proceedings of the IEEE Global Internet Symposium (GI)*, 2000.
- [27] I. Stoica, T. Ng, and H. Zhang. REUNITE: A recursive unicast approach to multicast. In *Proceedings of IEEE INFOCOM*, Mar. 2000.
- [28] D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. In *Proceedings of IEEE INFOCOM*, Mar. 2000.
- [29] J. Tian and G. Neufeld. Forwarding state reduction for sparse mode multicast communications. In *Proceedings of IEEE INFOCOM*, Mar. 1998.
- [30] D.-N. Yang and W. Liao. Optimizing state allocation for multicast communications. In *Proceedings of IEEE INFOCOM*, Mar. 2004.
- [31] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of NOSSDAV*, June 2001.