

A Study of Group-Tree Matching in Large Scale Group Communications *

Jun-Hong Cui¹, Li Lao², Mario Gerla²
jcui@cse.uconn.edu, llao@cs.ucla.edu, gerla@cs.ucla.edu

¹ Computer Science & Engineering Department, University of Connecticut, Storrs, CT 06029

² Computer Science Department, University of California, Los Angeles, CA 90095

Keywords

multicast, state scalability, group-tree matching, aggregated multicast

Abstract

As a mechanism to support group communications, multicasting faces a serious state scalability problem when there are large numbers of groups in the network: lots of resources (e.g., memory to maintain group state information) and control overhead (e.g., multicast tree setup and maintenance) are required to manage the groups. Recently, an efficient solution called **aggregated multicast** is proposed [8]. In this approach, groups are assigned to proper trees and multiple groups can share one delivery tree. A key problem in aggregated multicast is group-tree matching (i.e., matching groups to trees). In this paper, we investigate this group-tree matching problem. We first formally define the problem, and formulate two versions of the problem: static and dynamic. Since the static version is the fundamental part of the problem, in this paper, we focus our efforts on this aspect. We analyze the static version problem and prove that it is NP-complete. To tackle this hard problem, we propose three algorithms: one optimal (using Integer Linear Programming, or ILP), one near-optimal (using Greedy method), and one pseudo-dynamic algorithm. Simulation studies are conducted to compare these three algorithms. Our results show that Greedy algorithm is a feasible solution and its performance is very close to the ILP optimal solution, while pseudo-dynamic algorithm is a good heuristic for many cases where Greedy does not work well. The study of group-tree matching in this paper not only provides solutions for the static version problem, but also paves the way for formally investigating the dynamic version problem.

1. INTRODUCTION

With the rapid development of the Internet, there are many emerging large scale multi-user applications, such as news/software distributions, distributed interactive simulation (DIS), network games, distributed virtual collaborations, teleconferencing, telemedicine, teleducation, and stock quotes distribution, etc. All these applications involve multi-point communications (or group communications); that is, delivering data from one or more sources to multiple re-

ceivers. 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 to receive the data.

Multicast can be implemented at different network protocol layers, such as network layer (i.e., IP multicast), and application layer (i.e., application-layer multicast). Multicast can also employ different delivery structures, such as trees (e.g., in IP multicast) and meshes (e.g., in Narada [3], an application-layer multicast protocol). In a tree delivery structure, each in-tree node maintains the forwarding state. Data packets are duplicated at fork nodes and forwarded only once over each link. Due to its resource efficiency, the tree structure is widely used in multicast protocols. In this paper, we only focus on multicasting with the tree delivery structure.

Since multicast employs the concept of group, no matter at what level multicast is implemented, each multicast group traditionally 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 the 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, 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.

Recently, the multicast state scalability problem has prompted many interesting research works: some schemes attempt to reduce forwarding state at non-branched in-tree nodes [16, 14, 5]; some other schemes try to achieve state reduction by forwarding state aggregation at individual in-tree nodes [12, 15]. However, these schemes only consider the resource aspect of the state scalability problem.

A recent proposed approach, called aggregated multicast [8], explores both the resource and control overhead issues. In this scheme, multiple 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 a group-tree matching (i.e., assigning groups to trees) procedure since proper trees should be found to deliver data for groups. To solve the state scalability problem, the objective of a group-tree matching algorithm is to minimize the resources and control overhead. In previous studies [2, 13, 7, 6], several

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

aggregated multicast protocols using heuristic online group-tree matching algorithms have been proposed; however, there is no formal analysis of the group-tree matching problem.

In this paper, we formally define the group-tree matching problem, and formulate two versions of the problem: Static Pre-Defined Tree version and Dynamic On-Line version. In the static version, we assume all the groups are known beforehand, i.e., we have the knowledge of the global group information. This case is useful for multicast tree pre-dimensioning based on long-term traffic measurement. 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. Since the static version is fundamental to the whole problem solving, we focus our efforts on the Static Pre-Defined Tree problem in this paper. We analyze the complexity of this problem and show that it is NP-complete. We propose three algorithms: one optimal (using Integer Linear Programming, or ILP), one near-optimal (using Greedy method), and one pseudo-dynamic algorithm. By simulation studies, we show that the Greedy method is much faster and less time-consuming than the ILP approach while the performance is not significantly sacrificed (less than 1.5% for most of the simulated cases). The pseudo-dynamic algorithm is even faster and more resource efficient, but it trades off performance for efficiency. The study of the group-tree matching problem in this paper not only provides solutions for the static version problem, but also paves the way for formally investigating the dynamic version problem.

The rest of this paper is organized as follows. In Section 2, we describe and formulate the group-tree matching problem. Then we present three algorithms for the static version problem, and give formal time complexity analysis of these algorithms in Section 3 and 4. After that, we conduct simulation studies and compare different algorithms quantitatively in Section 5. Finally, we give a brief summary and conclude the paper.

2. THE GROUP-TREE MATCHING PROBLEM

2.1 Problem Description

In traditional multicast, each group uses one delivery tree, while in aggregated multicast [8], multiple groups are forced to share one aggregated tree. Thus, to implement aggregated multicast, we need to match groups to aggregated trees, i.e., do group-tree matching. One thing to note is that aggregated multicast is targeted for a single domain, especially a transit domain. Thus, in this paper, we also consider the group-tree matching problem in a single domain.

Given a group and a tree, the set of group members (including sources and receivers) and the set of tree leaves are not always identical. If all the tree leaves have group members, the tree is called a *perfect match* for the group. If there are tree leaves that do not have group members, we call this tree a *leaky match* for the group. In this case, the tree is “bigger” than the group. In other words, if we deliver data for the group using the “leaky-matched” tree, we send data to parts of the tree with no 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,

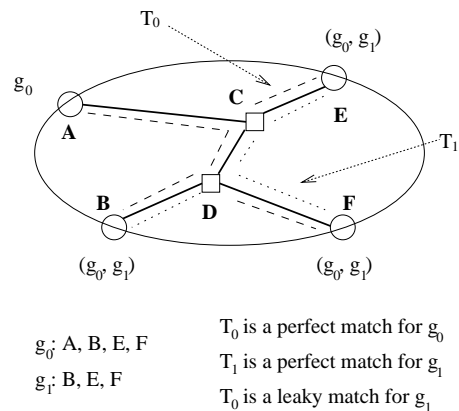


Figure 1: Examples of Perfect Match and Leaky Match

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 . 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 the nodes that are not members for the group. Therefore, we trade off bandwidth for state scalability.

Theoretically, the group-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-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 pre-defined 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

Network Model The target network is modelled as an undirected graph $G(V, E)$. Each edge (i, j) is assigned a positive cost $c_{ij} = c_{ji}$, which represents the cost to transport 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 that tree is

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

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

Bandwidth Waste Consider a network $G(V, E)$, in which a multicast routing algorithm A (for example, the shortest path tree algorithm) is used to set up multicast trees. Given a multicast group g , let $t^n(g)$ be the multicast tree computed by the routing algorithm (we refer to $t^n(g)$ as the “native” tree for group g). If this group is covered by an aggregated tree $t(g)$, its **bandwidth waste** ratio can be defined as

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

This metric directly reflects the relative bandwidth waste fraction when tree $t(g)$ is used to carry data for group g instead of the native tree $t^n(g)$. Then, the bandwidth waste can be quantified as $D(g) \times \delta(t, g) \times C(t^n(g))$ if the amount of data transmitted is $D(g)$. Following the assumption of equal link cost of 1, bandwidth waste ratio can be represented as

$$\delta(t, g) = \frac{|(t(g))| - |t^n(g)|}{|t^n(g)| - 1}. \quad (3)$$

To control the amount of bandwidth waste, in our group-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. 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, DVMRP/PIM-DM, PIM-SSM) or core-based tree (e.g., CBT and PIM-SM). 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.

Aggregation Degree Let N_{groups} be the number of multicast groups in the network and N_{trees} the number of aggregated trees used to cover those groups, then **aggregation degree** is defined as

$$AD = \frac{N_{groups}}{N_{trees}}. \quad (4)$$

AD is a direct measurement of aggregation: the bigger AD is, the smaller number of aggregated trees we need to manage, and thus the less requirement of the resources and control overhead. In one sentence, the bigger AD is, the better aggregation we could achieve. In fact, maximizing aggregation degree is the optimization goal in our group-tree matching algorithms.

2.3 Problem Formulation

Depending upon how aggregated multicast is deployed, we formulate the group-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 $G(V, E)$, a set of multicast groups $Grps$, a multicast routing algorithm A , and a bandwidth waste threshold bth . The goal is to find N trees (each of them covers 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 [4].

- **INSTANCE:** 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 $Grps$ 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 $Grps$ 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 $Grps$.
- 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 $Grps$ 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.

2.3.2 Dynamic On-Line Trees

The dynamic version of the group-tree matching problem is useful for on-line systems. In this case, instead of a static set of groups, groups dynamically join and leave. 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 starts, with the objective of minimizing the number of aggregated trees (i.e., maximizing the aggregation degree) without violating the given bandwidth waste threshold bth . Several previous studies [2, 13, 7, 6] proposed heuristic online group-tree matching algorithms, but there is no formal performance analysis.

In this paper, we mainly investigate the Static Pre-Defined Tree problem, as it is fundamental to the whole problem, and its solution could cast light on the formal study of the dynamic version problem. In the following sections, we propose three algorithms for the static NP-complete problem: one optimal algorithm using ILP, one near-optimal algorithm using greedy method, and one pseudo-dynamic algorithm.

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-tree mapping. The candidate tree generation and group-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, we describe the algorithms to solve each of the sub-problems, and then analyze the time complexity of these algorithms.

3.1 Candidate Tree Generation

The sub-problem of candidate tree generation can be formulated as follows: given a network graph $G(V, E)$, a multicast routing algorithm A used to set up multicast trees¹, and a set of groups $Grps$, for each multicast group $g \in Grps$, 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 $g \in Grps$, 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 $T(g)$. Then this native tree is extended to generate more potential trees with no more than l additional links², 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 not included in $t^n(g)$ is no more than l): if it is, then the tree can be extended to contain v . The generated trees are also included in $T(g)$. 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

¹In this paper, we use the shortest path tree algorithm as an example since many practical multicast routing algorithms use shortest path trees (e.g, MOSPF, DVMRP/PIM-DM, PIM-SSM) or core-based trees (e.g., CBT and PIM-SM), which are essentially shortest path trees rooted at the core.

²Since 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 algorithms, such as minimum spanning tree, candidate trees can not be simply obtained by tree extension.

dividing the set of multicast groups $Grps$ 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 $Grps$. In this way, we convert this problem to the MINIMUM SET COVER problem.

MINIMUM SET COVER is a classical NP-complete problem, and a large number of approximation algorithms have been proposed to solve this problem [10, 11]. Here, 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 that can be expressed in ILP formulation with limited variables and constraints can be solved optimally with free or commercially available softwares. Therefore, the first step 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:

$$\sum_{j=1}^{N_t} c_{ij} \geq 1 \quad \forall i \in [0, N_g]$$

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 the optimal solution 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 $Grps$, the candidate trees $T = \cup_{g \in Grps} T(g)$, and the mapping between each tree t and its covered groups $G^c(t)$, this algorithm iteratively finds the tree that covers the largest number of groups, adds it into the selected tree set T^s , and removes these groups from $Grps$. It repeats this step until $Grps$ is empty, which means all groups have been covered by at least one tree.

3.3 Group-Tree Mapping

Given the groups $Grps$ 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.

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

```
1:  $T^s \leftarrow \emptyset$ 
2: while  $Grps$  is not empty do
3:    $max\_cover \leftarrow 0$ 
4:    $best\_tree \leftarrow 0$ 
5:   for all  $t \in T$  do
6:     if  $|G^c(t)| > max\_cover$  then
7:        $max\_cover \leftarrow |G^c(t)|$ 
8:        $best\_tree \leftarrow t$ 
9:     end if
10:  end for
11:   $T^s \leftarrow T^s \cup best\_tree$ 
12:   $T \leftarrow T - best\_tree$ 
13:   $Grps \leftarrow Grps - G^c(best\_tree)$ 
14: end while
15: return  $T^s$ 
```

Table 1: Definition of variables in the complexity analysis

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 ($N_{ts} \leq N_g$)

3.4 Complexity Analysis

Clearly, the time complexity of the ILP algorithm is not polynomial since the MINIMUM SET COVER problem is NP-complete. Hence, we focus on the complexity analysis of the Greedy algorithm. To facilitate our analysis, we define several variables as shown in Table 1.

In the candidate tree generation algorithm, given a tree t , finding all the trees that are extended from t requires scanning every non-tree node v and deciding if the extended shortest path tree satisfies the bandwidth waste threshold requirement, which takes $O(\binom{m}{1} m^2) = O(m^3)$ time. Since adding one node introduces at least one additional link, we can derive that the operation of finding trees extended from t with additional l links involves at most l times of node scanning, which takes $O(\binom{m}{l} m^2) = O(\frac{m!}{(m-l)!} m^2)$. Therefore, the time complexity of finding the extended trees with no more than l links is:

$$O\left(\sum_{i=1}^l \frac{m!}{(m-i)!} m^2\right) \ll O\left(\sum_{i=0}^n \frac{m!}{(m-i)!} m^2\right) = O(2^m)$$

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

$$O\left(m^3 + \frac{m(m-1)}{2} m^2 + \frac{m(m-1)(m-2)}{6} m^2\right) = O(m^5)$$

In this way, the candidate tree selection algorithm takes $O(N_g m^5)$ 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: $N_{tc} = O(N_g m^5)$.

For the tree selection sub-problem, the Greedy algorithm requires scanning all the candidate trees for N_{ts} times un-

til N_{ts} trees are selected. Obviously, $N_{ts} \leq N_g$, so the time complexity of this algorithm is $O(N_{tc} N_{ts}) = O((N_g m^5) N_g) = O(N_g^2 m^5)$.

Finally, in the group-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. To solve this issue, we propose a heuristic Pseudo-Dynamic algorithm in next section.

4. PSEUDO-DYNAMIC ALGORITHM FOR STATIC PRE-DEFINED TREES

We present a so-called Pseudo-Dynamic algorithm for the Static Pre-Defined Tree problem in this section.

The basic idea is as follows. Given a static set of groups $Grps$, we randomly pick groups and let them join the network one by one. In this way, we actually have a dynamic group trace, for which we can use a dynamic on-line algorithm. For completeness, we briefly review a heuristic algorithm used by some previous studies [7, 6] as follows. Note that, in our proposed Pseudo-Dynamic algorithm, there is only group join and no leave (this is why the algorithm is called Pseudo-Dynamic); thus, we only describe the group join procedure.

When a new group g joins the network, we first identify if there are eligible existing trees to cover it. For each existing tree t , if it can cover g without exceeding bth , then t is a candidate tree. Otherwise, it is extended to cover g , and if the resulting extended tree t^e satisfies the bandwidth waste requirement for all of the groups mapped to t , i.e., $G^c(t) \cup g$, then t^e is considered as a candidate tree. Note that, when extending an existing tree, we must check if the extended tree still satisfies the requirement of bth for the groups mapped to the original tree to ensure the bound on the bandwidth waste is valid. If there are more than one candidate trees for group g , then the one with the minimum bandwidth waste is selected as the final tree $t^f(g)$; otherwise, the native tree $t^n(g)$ of this group is constructed as its final tree.

In our Pseudo-Dynamic algorithm, by running the above dynamic procedure, we actually obtain the proper trees covering all the groups after all of them join the network. Obviously, the result may be far from optimal, and can be affected by the order of group join. We can improve the performance by running the algorithm on multiple group traces with different group join orders. As a matter of fact, in our simulation studies, we will show that our 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 of the Pseudo-Dynamic algorithm is $O(N_g N_{ts} n) = O(N_g^2 n) = O(N_g^2 m^2)$, which is much faster than $O(N_g^2 m^5)$ in the case of the Greedy algorithm.

5. SIMULATION STUDIES

We have implemented the proposed algorithms and tested

them under various conditions. In this section, we first introduce the simulation environments and performance metrics, and then we present the simulation results.

5.1 Simulation Settings

In our simulations, we use a network topology abstracted from the AT&T IP backbone, a real network topology with 123 nodes. This network consists of 9 gateway routers, 9 backbone routers, 9 remote GSR (Gigabit Switch Routers) access routers, and 96 remote access routers. We conduct an abstraction procedure to generate a simplified network of 54 nodes as follows: (1) the gateway routers and backbone routers are kept as is; (2) the remote access routers attached to the same gateway or backbone router are “contracted” into one *contracted node*; and (3) one additional *exchange node* is created for each gateway router to represent a peering network or a public exchange point to which the gateway router is connected.

In the obtained network topology, we generate group instances to run different group-tree matching algorithms. We use *Random Node Weight Model* [9] 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. In our simulated network, the gateway nodes and backbone nodes are assumed to be core routers only and are assigned a weight of 0. The weight of a contracted node is related to the number of access routers it represents. The exchange nodes are each assigned a weight of 0.9, since they usually connects to peering networks with a large number of group members.

To well control the number of groups, though we aim to evaluate the algorithms for the static version of the group-tree matching problem, we obtain the groups in the following dynamic procedure. We assume the multicast group requests arrive as a Poisson process, and the lifetime of a group follows an exponential distribution. After the network reaches the steady state, we know the average number of groups. 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 number of groups in the network. The group-tree matching algorithms for the Static Pre-Defined Tree problem are executed on the active groups every 10 seconds after 400 seconds when the steady state is usually reached. In this way, we collect 20 sampling points in each simulation. At various sampling points, we obtain different sets of groups with the same average number of groups. We measure the performance metrics for each set of groups, and then compute the averages as the final results.

To evaluate the performance of the group-tree mapping algorithms, besides Aggregation Degree (AD) (introduced earlier), we define the following metrics:

State Reduction Ratio (SRR): Since the 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: $SRR = 1 - \frac{S_{agg}}{S_{no-agg}}$, where S_{agg} and S_{no-agg} represent 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 computation overhead (and sometimes memory) required for the algorithm.

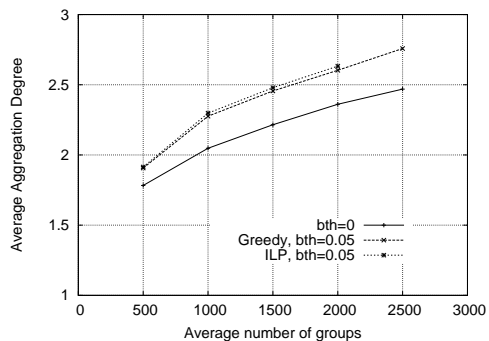


Figure 2: AD of ILP and Greedy algorithms vs. number of concurrently active groups.

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

5.2 Results and Analysis

5.2.1 Comparing ILP with Greedy

We first compare the performance of the ILP and Greedy algorithms by varying the bandwidth waste threshold (*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 are unable to obtain any results when the bandwidth waste threshold is very high or the number of groups is very large.

Figure 2 plots the results of AD (Aggregation Degree) for the 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 the native trees. As bth increases, ILP out-performs Greedy. This 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 improvement of ILP vs. 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, respectively, which means ILP require approximately 9 trees less than Greedy on average.

Figure 3 compares SRR (State Reduction Ratio) of the ILP and Greedy algorithms, and it exhibits a similar trend as Figure 2. Therefore, we can conclude that Greedy is able to achieve near-optimal performance in reducing the number of aggregated trees and multicast state entries.

Even though ILP achieves slightly higher performance than Greedy, this gain comes at the cost of additional computation overhead. To demonstrate this, we show the PET (Program Execution Time) ratio of ILP and Greedy in Figure 4. We observe that for $bth = 0$, the PET ratio increases approximately linearly with the number of co-existing groups. Due to the use of log-scale for the y axis, the linearity corresponds to an exponential relationship between the time ratio and the number of groups. For $bth = 0.05$, the ratio increases even

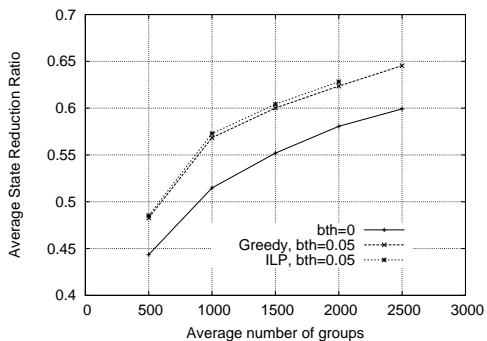


Figure 3: SRR of ILP and Greedy algorithms vs. number of concurrently active groups.

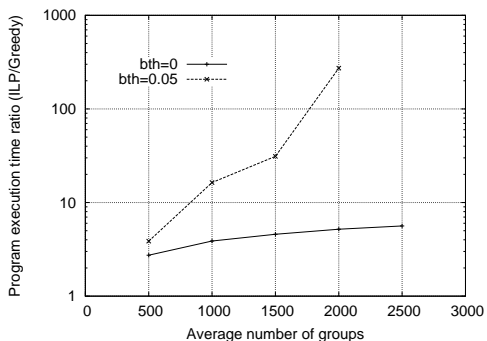


Figure 4: PET Ratio of ILP and Greedy algorithms vs. number of concurrently active groups.

more rapidly. This indicates that the computational cost of ILP becomes 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.

5.2.2 Greedy vs. Pseudo-Dynamic

As shown in the previous section, the ILP algorithm is not feasible for large numbers of groups and high bandwidth waste threshold, so we now only compare the performance of Greedy and Pseudo-Dynamic algorithms. We expect that the latter algorithm is not able to perform as well as the former, since it does not exploit all the potential trees for a group and select trees based on a global view of the relationship between groups and trees as Greedy does. Figure 5 and 6 plot the AD and SRR, respectively, 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

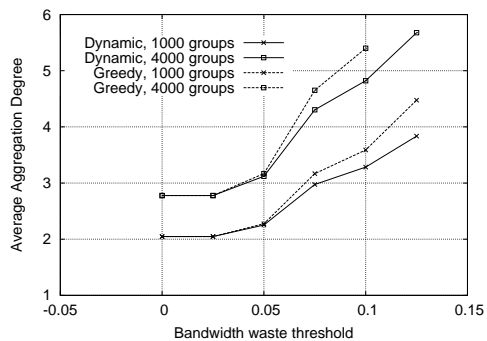


Figure 5: AD of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold.

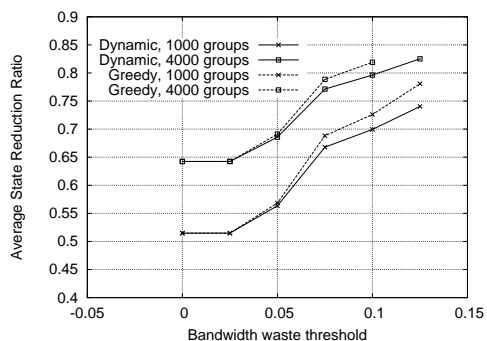


Figure 6: SRR of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold.

bandwidth waste threshold increases, the difference between the two algorithms becomes more distinct. We want to point out here that, in comparison with the Greedy algorithm), the performance penalty resulting from using the Pseudo-Dynamic algorithm is very small: under the conditions tested, in the worst case, the performance reduction by the Pseudo-Dynamic algorithm is 14.3% for AD and only 5.16% for SRR, relative to the Greedy algorithm.

Finally, we plot the PET ratio of the Greedy and Pseudo-Dynamic algorithms in Figure 7. 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 7, 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 7, 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 we can see, for 4000 groups, when $bth \geq 0.075$, the Pseudo-Dynamic algorithm starts to

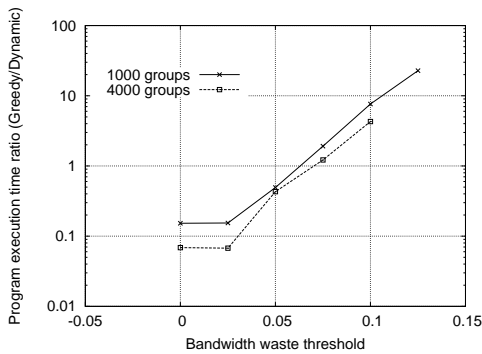


Figure 7: PET Ratio of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold.

outperform the Greedy algorithm.

5.2.3 Summary

In summary, our simulation studies have shown the following results: 1) Among the algorithms for Static Pre-Defined Trees, the performance of multicast state scalability degrades in the order of ILP (optimum), Greedy (near-optimum), and Pseudo-Dynamic; 2) The ILP algorithm is not feasible in reality due to its significant processing overhead and rigorous memory requirement; 3) The Greedy algorithm has a better trade-off between performance and efficiency, and it is very effective when the bandwidth waste threshold is small; 4) The Pseudo-Dynamic algorithm is the fastest one among the three, but it sacrifices performance significantly. However, when the bandwidth waste threshold is very big, it becomes a practical solution compared with the Greedy algorithm.

6. CONCLUSIONS

In this paper, we have studied the group-tree matching problem in large scale group communications. We formulate two versions of the problem: static version and dynamic version. For the static version of the problem, we prove that it is NP-complete, and propose three algorithms (i.e., ILP, Greedy, and Pseudo-Dynamic). By simulation studies, we find that the Greedy algorithm is a feasible solution to the Static Pre-Defined Tree problem when the bandwidth waste threshold (bth) is small, and its performance is very close 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 (with 14.3% in the worst case of our simulations).

Though the focus of this paper is the study of the static version of the group-tree matching problem, we believe we have set up a prelude for the formal analysis of the dynamic version problem. Both the Greedy and Pseudo-Dynamic algorithms could serve as a good starting point for the upper bound analysis of the dynamic on-line group-tree matching problem.

7. REFERENCES

- [1] *lp_solve, Version 5.0.*
http://groups.yahoo.com/group/lp_solve/.
- [2] F. Y. Y. Cheng and R. K. C. Chang. A tree switching protocol for multicast state reduction. In *Proceedings*

- of *IEEE Symposium on Computers and Communications (ISCC'00)*, 2000.
- [3] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. *Proceedings of ACM Sigmetrics*, June 2000.
- [4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [5] L. H. M. Costa, S. Fdida, and O. C. M. Duarte. Hop-by-hop multicast routing protocol. *Proceedings of SIGCOMM'01*, Aug. 2001.
- [6] 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.
- [7] 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.
- [8] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast: an approach to reduce multicast state. *Proceedings of Sixth Global Internet Symposium (GI2001)*, Nov. 2001.
- [9] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast with inter-group tree sharing. *Proceedings of NGC2001*, Nov. 2001.
- [10] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.
- [11] V. T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, June 1997.
- [12] 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.
- [13] S. Song, Z.-L. Zhang, B.-Y. Choi, and D. H. Du. Protocol independent multicast group aggregation scheme for the global area multicast. In *In the Proceedings of the IEEE Global Internet Symposium (Globecom'00)*, 2000.
- [14] I. Stoica, T. Ng, and H. Zhang. REUNITE: A recursive unicast approach to multicast. In *Proceedings of IEEE INFOCOM'00*, Tel Aviv, Israel, Mar. 2000.
- [15] D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. *Proceedings of IEEE INFOCOM*, Mar. 2000.
- [16] J. Tian and G. Neufeld. Forwarding state reduction for sparse mode multicast communications. *Proceedings of IEEE INFOCOM*, Mar. 1998.