Tackling Group-Tree Matching in Large Scale Group Communications

Li Lao[1], Jun-Hong Cui[2], Mario Gerla[1]
llao@cs.ucla.edu, jcui@cse.uconn.edu, gerla@cs.ucla.edu
[1] Computer Science Department, University of California, Los Angeles, CA 90095
[2] Computer Science & Engineering Department, University of Connecticut, Storrs, CT 06029

Technical Report TR040022
06-29-2004
Computer Science Department
UCLA

***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. We analyze the static version of the problem 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 general heuristic on-line group-tree matching algorithm. Simulation studies are conducted to compare the three algorithms for the static version. Our results show that Greedy algorithm is a feasible solution to the static problem and its performance is very close the ILP optimal solution, while pseudo-dynamic algorithm is a good heuristic for many cases where Greedy does not work well. We also evaluate the performance of the heuristic online algorithm, and show that it is a practical solution to the dynamic on-line group-tree matching problem.**

I. 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), distributed network games, distributed virtual collaborations, teleconferencing, telemedicine, teleeducation, 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 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 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 multicast). Multicast can also employ different delivery structures, such as tree (e.g., in IP multicast) and mesh (e.g., in Narada [3], an application multicast protocol). In a tree delivery structure, each in-tree node maintains the forwarding state, and data packets are duplicated at fork nodes and are forwarded only once over each link. Due to its resource efficiency, tree is widely used in multicast protocols. In this paper, we only focus on multicasting with 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 large amount of resources and management overhead will be involved. Hence, the network performance will be tremendously degraded. This issue is referred as multicast state scalability problem. It will be exacerbated with the increasing demand of the multi-user applications.

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

A recent proposed approach, called aggregated multicast [8], 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-tree matching (i.e., assigning groups to trees) procedure since proper trees should be found to deliver data for the groups. To solve the state scalability problem, the objective of the group-tree matching algorithms would be to minimize the resources and control overhead. In previous studies [2], [13], [7], [6], several 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, and there is no formal evaluation of how good the online heuristics are.

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. We analyze the complexity of the static version of the problem and show that it is NP-complete. We propose three algorithms for this problem: one optimal (using Linear Integer Programming, or ILP), one near-optimal (using Greedy method), and one pseudo-dynamic algorithm. By simulation studies, we show that Greedy method is much faster and less time consuming than ILP while the performance is not significantly sacrificed (less than 1.5% for most of the simulated cases). Pseudo-dynamic algorithm is even faster and more resource efficient, but we trade 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 online systems. We present a general online heuristic group-tree matching algorithm, and evaluate its performance by comparing with its upper bound, obtained using the static algorithm. We find that the dynamic on-line algorithm is a practical solution to the dynamic group-tree matching problem with limited performance penalty and reasonable computation requirement.

The rest of this paper is organized as follows. In Section II, we first describe and formulate the group-tree matching problem. Then we present the algorithms for both the static and dynamic versions of the problem, and give formal time complexity analysis of these algorithms (Section III, IV and V). After that, we conduct simulation studies and compare different algorithms quantitatively in Section VI. Finally, we give a brief summary and conclude the paper.

II. The Group-Tree Matching Problem

A. 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 implem1ent aggregated multicast, we need to match groups to aggregated trees, i.e., do group-tree matching.

Given a group and a tree, the set of the group members (sources and receivers) and the tree leaves are not always identical. If all the tree leaves have the group members, the tree is called a perfect match for the group. If there are leaves of the tree that do not have the 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, we send data to parts of the tree with no receivers. Some simple examples of perfect match and leaky match are illustrated

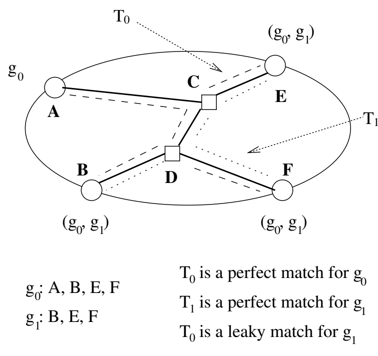
T_0
(g_0, g_1)
g_0
C
E
A
T_1
B
D
F
(g_0, g_1)
(g_0, g_1)
g_0: A, B, E, F
g_1: B, E, F
T_0 is a perfect match for g_0
T_1 is a perfect match for g_1
T_0 is a leaky match for g_1

Fig. 1. Examples of Perfect Match and Leaky Match

in Figure 1. Clearly, by leaky match, we can achieve better aggregation (i.e., using less trees to cover more groups). A disadvantage of the leaky match is that some bandwidth is wasted to deliver data to nodes that are not members for the group. Therefore, we trade off bandwidth for state scalability. Theoretically, group-tree matching problem is an intractable multi-objective (minimizing bandwidth waste and maximizing aggregation) optimization problem, with the two objectives self-contradicting each other. In reality, however, a network manager could seek efficient group-tree matching algorithms which can achieve best aggregation while keeping bandwidth waste under some given threshold. In this way, the problem becomes simplified.

B. Network Model and Definitions

1) Network Model: The 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, total cost to distribute a unit amount of data over that tree is

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

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

2) Bandwidth Waste: Now consider a network $G(V, E)$, in which multicast routing algorithm A (for example, shortest path tree algorithm) is used to setup multicast trees. Given a multicast group g, let $t^n(g)$ be the multicast tree computed by the routing algorithm (we refer $t^n(g)$ as the "native" tree for group g). Alternatively, this group can be covered by an aggregated tree $t(g)$, and its **bandwidth waste** ratio is defined as

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

This metric directly reflects bandwidth waste ratio 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

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

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), and 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.

3) 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, **aggregation degree** is defined as

$$AD = \frac{N_{groups}}{N_{trees}}. \tag{4}$$

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

C. Problem Formulation

Depending upon how aggregated multicast is used, 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.

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 node set) 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 given threshold bth. This is the problem we need to solve to build a set of pre-defined aggregated trees based on long-term traffic measurement information.

In fact, we can show that 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 $C\prime \subseteq C$ such that every element in S belongs to at least one member of $C\prime$.
- MEASURE: Cardinality of the set cover, i.e., $|C\prime|$.

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 bandwidth waste threshold bth given size k. Suppose we are given k trees, to validate if these trees can cover 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 of this problem is actually a MINIMUM SET COVER problem. We specify the special case as follows: target trees can only be selected from the set of native trees of $Grps$ instead of all possible trees in the network. Now, we can present an instance of this problem as an instance of MINIMUM SET COVER problem:

- Let the multicast group set $Grps$ be S.
- Let the native tree set of the groups denoted by T^n. For each tree $t \in T^n$, we represent the group set covered by t with bandwidth waste below bth as $G^c(t)$. We denote the collection of $G^c(t)$ as $G^c(T^n)$.
- Let the collection $G^c(T^n)$ be C.

Then the objective of finding minimum number of trees T^m to cover the groups $Grps$ becomes minimizing the cardinality of set $C\prime$, where $C\prime \subseteq C$ such that every element in S belongs to at least one member of $C\prime$. 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 harder than this special case, thus it 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 III and V respectively.

2) Dynamic On-Line Trees: The dynamic version of the group-tree matching problem is more meaningful for practical purposes. In this case, instead of a static set of groups, groups dynamically join and leave. Our 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 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 upper bound analysis to evaluate the performance of these algorithms. In Section IV, we will first describe a general on-line dynamic algorithm, and then compare its performance with the upper bound.

III. Algorithms for Static Pre-Defined Trees

In this section, we present two algorithms for Static Pre-Defined Trees. The basic idea behind these algorithms is to first find the candidate trees that can be used to cover a subset of given multicast groups, convert the problem of selecting the minimum number of trees for all 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 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.

A. 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 trees, 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 links, 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 edge $e \in E$ whether e is adjacent to t: if it is, then the tree can be extended to contain e. 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.

B. 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 $Grps$ 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 MINIMUM SET COVER problem.

MINIMUM SET COVER problem 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.

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, our major task is to develop ILP formulation for tree selection problem. Before presenting the ILP formulation, we define the following variables:

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

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

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:

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

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.

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 $T = \cup_{\forall g \in Grps} T(g)$, 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 remove these groups from $Grps$, until $Grps$ is empty, which means all groups have been covered by at least one tree.

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 $T^s$
```

C. 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 minimum bandwidth waste as the final tree $t^f(g)$ for this group.

D. 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 I.

In candidate tree generation algorithm, given a tree t, finding all trees that are extended from t with one additional link

TABLE I
DEFINITION OF VARIABLES IN COMPLEXITY ANALYSIS

m	number of nodes
n	number of links
N_g	number of groups
N_{t^c}	number of candidate trees
N_{t^s}	number of selected trees ($N_{t^s} \leq N_g$)

requires scanning every link $e \in E$ and deciding if it is adjacent to t, which takes $O(\binom{n}{1}) = O(n)$ time. Similarly, we can derive that the operation of finding trees extended from t with additional l links takes $O(\binom{n}{l}) = O(\frac{n!}{(n-l)!l!})$. Therefore, the time complexity of finding the extended trees with no more than l links is:

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

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 numbers in order to control bandwidth waste. If we assume $l \leq 3$, the time complexity of generating candidate trees for one group becomes:

$$O(n + \frac{n(n-1)}{2} + \frac{n(n-1)(n-2)}{6}) = O(n^3)$$

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

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

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 $O(N_g N_{t^s} n) = O({N_g}^2 n)$ 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 time.

What to do when bth is big? Obviously, when bandwidth waste threshold bth is big, the Greedy algorithm will become time consuming. Inspired by the dynamic on-line group tree matching heuristics, we propose a pseudo-dynamic algorithm for Static Pre-Defined Tree problem, and it will be presented in Section V.

IV. Dynamic On-Line Algorithm

The algorithms discussed above are expected to achieve optimal or near-optimal performance in minimizing the number of aggregated trees without violating the bandwidth waste threshold. However, they are used to match a static set of groups onto trees, and thus cannot be used as "on-line" algorithms. To solve this problem, many heuristics that determines group-tree matching as groups join and leave the network have been proposed [2], [13], [7], [6]. In this section, we briefly present a general dynamic on-line algorithm.

Algorithm 2 shows the procedure for group join. When a new group g joins the network, we first identify if there are eligible existing trees to cover it. For each existing tree $t \in T^s$, 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 extended tree still satisfies the requirements of bth for 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 of this group is constructed as its final tree. When a group leaves the network, a similar procedure will be activated. However, some trees will be shrunk instead of extended in order to obtain better set of trees which can cover the active groups.

Algorithm 2 Dynamic-Join(g)

```
1:  $T^c(g) \leftarrow null$ //initialize candidate tree set
2:  for all $t \in T$ do
3:    //$T$ is the existing tree set
4:    if $t$ covers $g$ then
5:      compute the bandwidth waste $\delta(t, g)$
6:      if $\delta(t, g) < bth$ then
7:        $T^c(g) = T^c(g) \cup t$
8:      end if
9:    else
10:     extend tree $t$ to $t^e$ to cover group $g$
11:     compute the maximum bandwidth waste $\delta_{max}$ between $t^e$ and its covered groups including $g$
12:     if $\delta_{max} < bth$ then
13:       $T^c(g) = T^c(g) \cup t^e$
14:     end if
15:   end if
16: end for
17: if $T^c(g) == null$ then
18:   the native tree $t^n(g)$ is used as $t^f(g)$
19: else
20:   choose a tree with min. bandwidth waste as $t^f(g)$
21: end if
```

In the Dynamic On-Line algorithm, the existing trees T are searched to find a best tree for each group. Checking the eligibility and bandwidth waste threshold of an existing tree or an extended tree requires $O(n)$ time, and finding the native tree when no existing tree is appropriate requires $O(n)$ time, so the time complexity for group join is $O(|T|n)$, while $|T|$ is bounded by the number of existing groups in the network. A similar analysis applies to group leave.

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. In fact, we can obtain an upper bound for each of the sample points in the group dynamics as follows: at a given sample point, we collect a set of (currently active) groups, run a Static Pre-Defined Tree algorithm, and decide what is the minimum set of trees to cover these groups. In our simulation studies, we will investigate the performance of Dynamic On-Line algorithm by comparing the optimal or near-optimal solutions.

V. Pseudo-Dynamic Algorithm for Static Pre-Defined Trees

In Section III, we presented two algorithms: ILP algorithm, and Greedy algorithm. Though the latter one is practical when bandwidth waste threshold bth is small, we remark that the efficiency of 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 such called Pseudo-Dynamic Algorithm for the Static Pre-Defined Tree problem.

The basic idea is following: 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 Dynamic On-Line algorithm. Since in the generated group trace, there is only group join and no leave, when all the groups join the network, by running Dynamic On-Line algorithm, we actually obtain the proper trees covering all the groups. 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 studies, we will show that this Pseudo-Dynamic algorithm is very effective: much faster than Greedy algorithm when bth is big without sacrificing too much performance.

Following the previous time analysis, it is easy to see the time complexity for Pseudo-Dynamic algorithm $O(N_g N_{t^s} n) = O({N_g}^2 n)$, which is much faster than $O({N_g}^2 n^3)$ in the case of the Greedy algorithm.

VI. Simulation Studies

We 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.

A. Simulation Settings

In our simulations, we use a network topology abstracted from 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 the peering networks and Internet public exchange points 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, gateway nodes and backbone nodes are assumed to be core routers only and are assigned a weight of 0. The weights of the contracted nodes is related to the number of access routers it represents. Exchange nodes are each assigned a weight of 0.9, since they usually connects to peering networks with a large number of group members.

We assume the multicast group requests arrive as a Poisson process, and groups's lifetime follows an exponential distribution. After the network reaches steady state, we know the average number of groups. In our simulations, we fix average group life time as 100 seconds and simulation time as 600 seconds, and vary the group arrival rates to control the number of groups in the network. The group-tree matching algorithms for Static Pre-Defined Trees are executed on the active groups every 10 seconds, and the Dynamic On-Line algorithm is executed continuously as group joins and leaves the network, both after 400 seconds simulation time when steady state is reached.

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 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:

$$SRR = 1 - \frac{S_{agg}}{S_{no\_agg}} \tag{5}$$

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 is 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.

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 problems. An Intel Xeon 2.40GHz computer with 1GB memory is used to execute the simulation experiments.

B. Results and Analysis

1) ILP vs. Greedy: We first compare the performance of 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 ILP algorithm, we were unable to obtain results for higher bandwidth waste threshold and larger number of groups.

Figure 2 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

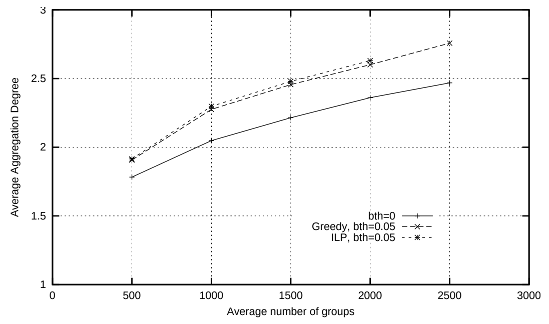

Fig. 2. AD of ILP and Greedy algorithms vs. number of concurrently active groups.

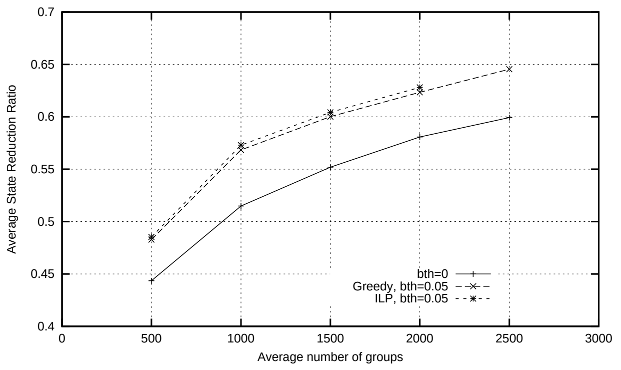

Fig. 3. SRR of ILP and Greedy algorithms vs. number of concurrently active groups.

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 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 ILP vs. Greedy algorithms, which 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 aggregation trees and multicast state entries.

Even though ILP achieves slightly higher performance than Greedy, this gain comes at a cost of additional computation overhead. To demonstrate this, we show the PET (Program Execution Time) ratio of ILP vs. Greedy in Figure 4. We observe that for $bth = 0$, the PET ratio increases approximately linearly with the number of co-existing group, which corresponds to an exponential relationship between the time ratio and the number of groups due to the use of log-scale for y axis; for $bth = 0.05$, the ratio increases even more rapidly. This indicates that computation cost of ILP will become too expensive when there are a large number of simultaneously

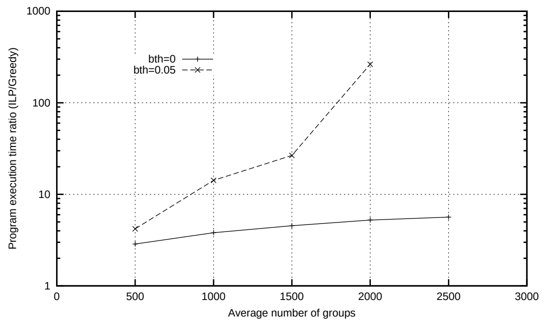

Fig. 4. PET Ratio of ILP and Greedy algorithms vs. number of concurrently active groups.

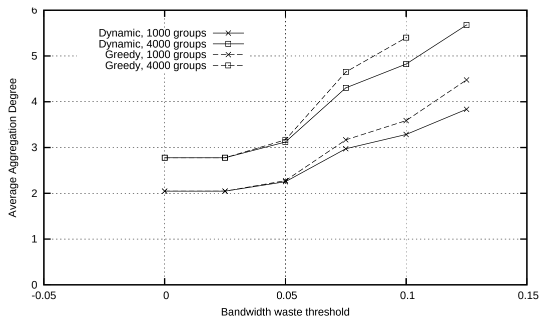

Fig. 5. AD of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold.

active groups and/or when the bandwidth waste threshold is high. For example, when $bth = 0.05$ and number of groups is 2500, the program is not able to finish within one day for ILP algorithm, whereas it takes only less than 50 seconds for the Greedy algorithm to complete.

2) Greedy vs. Pseudo-Dynamic: As shown in previous section, ILP algorithm is not feasible for large number of groups and high bandwidth waste threshold, so we now only compare the performance of Greedy vs. 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 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 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 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 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 algorithm is very small: under the conditions tested, in the worst case, the performance reduction by Pseudo-Dynamic algorithm is 14.3% for AD and only 5.16% for SRR, relative to Greedy algorithm.

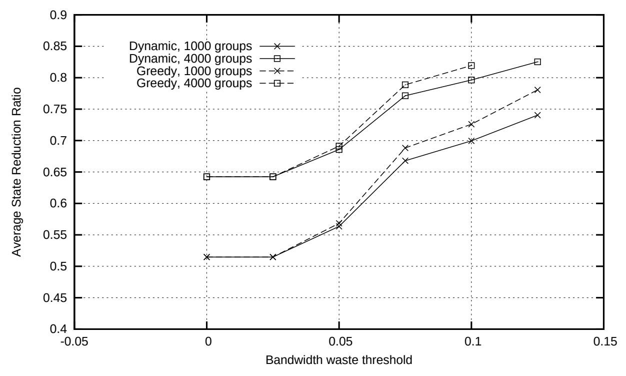

Fig. 6. SRR of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold.

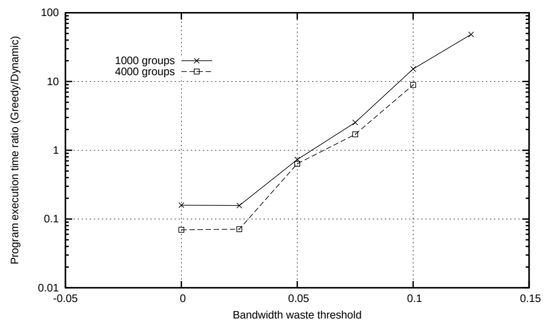

Fig. 7. PET Ratio of Greedy and Pseudo-Dynamic algorithms vs. bandwidth waste threshold.

Finally, we plot the PET ratio of Greedy vs. Pseudo-Dynamic algorithms in Figure 7. We found in our simulations that the PET of 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, Greedy algorithm runs faster; when $bth \geq 0.125$, Pseudo-Dynamic algorithm out-performs Greedy. When $bth = 0.125$, for 1000 groups, the PET ratio is as high as 48.3; for more than 2000 groups, the Greedy algorithm runs out of memory due to the storage requirement of a large number of potential trees for all groups, which is another disadvantages of ILP and Greedy algorithms.

3) Greedy vs. Dynamic On-Line: For on-line group-tree matching problem, Greedy algorithm gives an upper bound of how well Dynamic algorithm can solve this problem, because Dynamic algorithm incrementally finds group-tree matching for each group as groups join and leave the network by considering only existing trees as candidate trees, while Greedy algorithm takes advantage of the knowledge of groups joining later than the current one and tries to use one tree to include as many groups as possible. As illustrated in Figure 8 and 9, the results match our conjecture. In addition, in the case of AD, as bth increases, 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, as is true in reality: we do not want to waste too much bandwidth to improve state scalability.

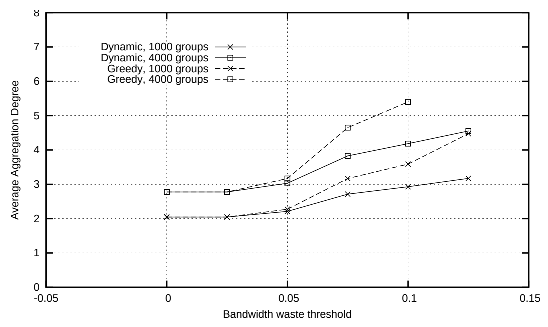

Fig. 8. AD of Greedy and Dynamic On-Line algorithms vs. bandwidth waste threshold.

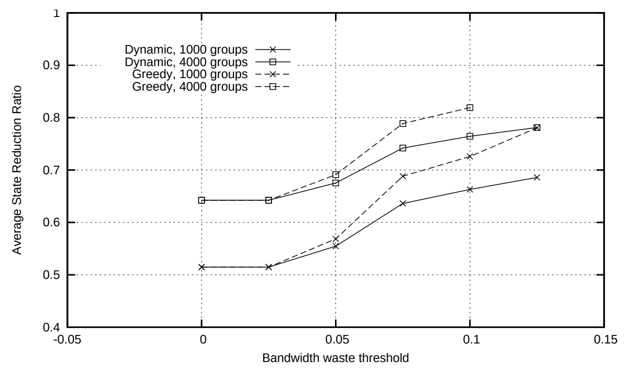

Fig. 9. SRR of Greedy and Dynamic On-Line algorithms vs. bandwidth waste threshold.

We also collected the execution time ratio of Greedy vs. Dynamic algorithm in 10. We observe again that Greedy algorithm generally takes longer time, especially for large bth, due to the exhaustive search of candidate trees required by the algorithm.

In conclusion, our simulation studies show 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; however, the first two algorithms are not feasible in reality due to significant processing and memory overhead. 2) The Dynamic On-Line algorithm is demonstrated to be a practical solution to the dynamic group-tree matching problem with limited performance penalty and reasonable computation requirement.

VII. Conclusions

In this paper, we have studied the group-tree matching problem in large scale group communications. We formulated two versions of the problem: static version and dynamic version. We proved that the static version of the problem is NP-complete. For each version of the problem, we proposed different solutions: three algorithms (ILP, Greedy, and Pseudo-Dynamic) are deigned for Static Pre-Defined Trees; and one general heuristic on-line algorithm is presented to solve the Dynamic On-Line Tree problem. By simulation studies, we find that Greedy algorithm is a feasible solution to the Static Pre-Defined Tree problem when bandwidth waste threshold

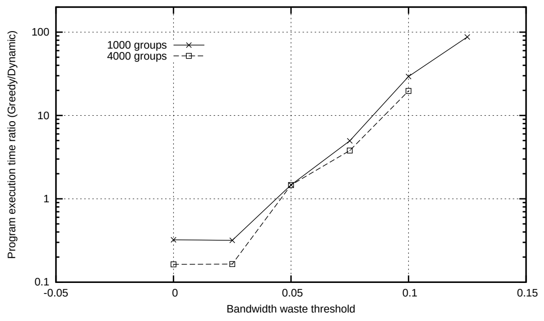

Fig. 10. PET ratio of Greedy and Dynamic On-Line algorithms vs. bandwidth waste threshold.

(bth) is small, and its performance is very close the ILP optimal solution (less than 1.5% for most of the cases). 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). We also formally evaluated the performance of dynamic on-line group-tree matching algorithm, and our experiments results demonstrate that the dynamic on-line algorithm is practical to solve on-line group-tree matching problem.

References

[1] *lp_solve, Version 5.0*. http://groups.yahoo.com/group/lp_solve/.

[2] Filli Y. Y. Cheng and Rocky K. C. Chang. A tree switching protocol for multicast state reduction. In *In the 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] Lus Henrique M.K. Costa, Serge Fdida, and Otto Carlos M.B. Duarte. Hop-by-hop multicast routing protocol. *Proceddings of SIGCOMM'01*, August 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] Jun-Hong Cui, Dario Maggiorini, Jinkyu Kim, Khaled Boussetta, and Mario Gerla. A protocol to improve the state scalability of source specific multicast. In *Proceedings of IEEE GLOBECOM*, November 2002.

[8] Aiguo Fei, Jun-Hong Cui, Mario Gerla, and Michalis Faloutsos. Aggregated Multicast: an approach to reduce multicast state. *Proceedings of Sixth Global Internet Symposium(GI2001)*, November 2001.

[9] Aiguo Fei, Jun-Hong Cui, Mario Gerla, and Michalis Faloutsos. Aggregated Multicast with inter-group tree sharing. *Proceedings of NGC2001*, November 2001.

[10] D. S. Johonson. 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] Sejun Song, Zhi-Li Zhang, Baek-Young Choi, and David H.C. 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.S. Ng, and H. Zhang. REUNITE: A recursive unicast approach to multicast. In *Proceedings of IEEE INFOCOM'00*, Tel Aviv, Israel, March 2000.

[15] D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. *Proceedings of IEEE INFOCOM*, March 2000.

[16] J. Tian and G. Neufeld. Forwarding state reduction for sparse mode multicast communications. *Proceedings of IEEE INFOCOM*, March 1998.