

# Dynamic On-line Group-Tree Matching for Large Scale Group Communications: A Performance Study

Jun-Hong Cui<sup>1</sup>, Li Lao<sup>2</sup>, M. Y. Sanadidi<sup>2</sup>, Mario Gerla<sup>2</sup>

*jcui@cse.uconn.edu, llao@cs.ucla.edu, medy@cs.ucla.edu, gerla@cs.ucla.edu*

<sup>1</sup> Computer Science & Engineering Department, University of Connecticut, Storrs, CT 06269

<sup>2</sup> Computer Science Department, University of California, Los Angeles, CA 90095

## Abstract

*Traditional IP multicast faces a serious state scalability problem when there are a large number of groups in the network. Recently, a novel approach called **aggregated multicast** was proposed [6], in which multiple groups share one delivery tree. A key problem in aggregated multicast is group-tree matching (i.e., assigning groups to trees). In this paper, we formally study the dynamic version of the group-tree matching problem. We propose a generic dynamic on-line algorithm (GDOA) and provide an approach to determine the upper bound on its performance. We quantitatively compare the performance of GDOA and other existing on-line heuristics. Extensive simulations demonstrate that GDOA is a very practical solution with promising performance and reasonable computation requirement.*

## 1 Introduction

Over the years, the Internet has seen increasing demands of group communication applications, including distributed interactive simulation (DIS), net games, video/audio conferencing, etc. Multicast, an efficient mechanism for group communications, utilizes a tree structure to deliver data from sender(s) to a group of receivers, with packets only forked at branching nodes in the tree. However, this approach has been typically plagued by state scalability problem: since each router needs to keep a forwarding entry for each group, the state information in a router will explode when there are numerous groups in the network; and similarly, the control messages will explode due to the maintenance overhead of large numbers of multicast trees.

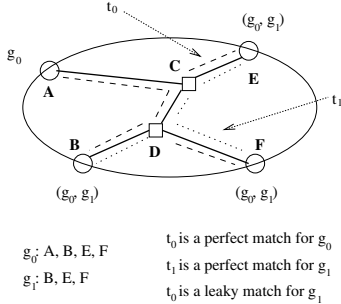
Recognition of the multicast state scalability problem has prompted many interesting research works.

Some approaches resort to different multicast architectures, such as application-layer multicast [7, 3] and Xcast [1], aiming to completely eliminate multicast state at routers. These solutions either sacrifice multicast efficiency or increase packet processing overhead at routers. Some other schemes reduce forwarding state at non-branched tree nodes [14, 12, 4, 15] or aggregate forwarding state at individual nodes [10, 13, 2]. However, all of them only consider reducing forwarding state at routers without solving the control overhead issue.

To tackle this state scalability problem, we recently proposed aggregated multicast [6], in which multiple multicast groups share a single *aggregated tree*. This way, the total number of trees may be significantly reduced; therefore, it solves both the forwarding state and control overhead issues. Note that a tree can not always cover a group “exactly”. If there are tree leaves that do not have group members, we actually deliver data to parts of the tree with no interested receivers and indeed waste some bandwidth. Thus, to assign groups to aggregated trees, a good group-tree matching algorithm is desired to minimize the resources and control overhead under a given bandwidth waste threshold.

We can formulate the group-tree matching problem into two versions: static and dynamic. In the static version, we have the global knowledge of all groups in advance. This case is useful for multicast pre-dimensioning based on long-term traffic measurement. For the dynamic version, groups dynamically join and leave, which is more meaningful for online systems and applications. In the literature, several studies [11, 5] have designed some dynamic online heuristics. However, there is neither formal analysis of how good these heuristics are compared with optimal solutions, nor performance comparison among these heuristics.

In this paper, we formally analyze the performance of dynamic on-line group-tree matching algorithms. We propose a generic algorithm GDOA and provide



**Figure 1. Perfect Match vs. Leaky Match.**

an approach to analyze the upper bound on its performance. By simulations, we evaluate GDOA and compare it with other existing on-line heuristics. We find that GDOA is a practical solution with promising performance and reasonable computation requirement.

The rest of this paper is organized as follows. In Section 2, we formulate the group-tree matching problem. In Section 3, we present GDOA and provide an upper-bound analysis. Then we present simulation results in Section 4 and conclude our paper in Section 5.

## 2 The Group-Tree Matching Problem

Unlike traditional multicast, aggregated multicast [6] forces multiple groups to share one aggregated tree. To assign groups to proper trees, we need to conduct group-tree matching.

### 2.1 Problem Description

For a group and a tree, if all tree leaves have group members, the tree is called a *perfect match* for the group. Otherwise, we actually send data to parts of the tree with no interested receivers, thus we call this tree a *leaky match*. We illustrate some simple examples of perfect match and leaky match in Figure 1.

By allowing an amount of “leaking”, we trade off bandwidth for state reduction, which is particularly useful as the number of groups scales up. Theoretically, group-tree matching problem is an intractable multi-objective (minimizing bandwidth waste and maximizing aggregation) optimization problem, with the two conflicting objectives. In reality, however, a network manager could seek efficient algorithms which achieve best aggregation while keeping bandwidth waste under some threshold. This is the main starting point of existing dynamic on-line heuristics [11, 5].

Before we present the formulation of the group-matching problem, we introduce some notations.

**Network Model** We model the network as an undirected graph  $G(V, E)$ . Without losing generality, we assume every link has a unit cost to transport unit traffic on the link. Given a multicast tree  $t$ , the total cost  $c(t)$  to distribute unit data over tree  $t$  is simply  $|t| - 1$ , where  $|t|$  denotes the number of nodes in  $t$ .

**Bandwidth Waste** Given a multicast group  $g$ , we assume a multicast tree  $t^n(g)$  (called the “native tree” of  $g$ ) can be obtained by a multicast routing algorithm  $A$  (e.g., shortest path tree algorithm) without aggregation. When this group is covered by an aggregated tree  $t(g)$ , the *bandwidth waste* is defined as

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

This metric reflects the relative additional bandwidth when tree  $t(g)$  instead of  $t^n(g)$  is used to carry data for group  $g$ . To control the amount of bandwidth waste, a group  $g$  is allowed to use a tree  $t$  only if  $\delta(t, g) \leq bth$ , where  $bth$  is a pre-defined bandwidth waste threshold.

**Aggregation Degree** Let  $N_g$  and  $N_t$  be the total number of multicast groups and aggregated trees respectively, the *aggregation degree* is defined as

$$AD = \frac{N_g}{N_t}. \quad (2)$$

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

### 2.2 Problem Formulation

As explained earlier, we can formulate the group-tree matching problem into two versions: static pre-defined version and dynamic on-line version.

**Static Pre-Defined Trees** In the static version, we are given: a network  $G(V, E)$ , multicast groups  $Grps$ , a multicast routing algorithm  $A$ , and a bandwidth waste threshold  $bth$ . The goal is to find  $N$  distinct trees and a matching from groups to trees such that every group is covered by a tree with the bandwidth waste below  $bth$ , with the objective of minimizing  $N$  (equivalent to maximizing  $AD$ ). We need to solve this problem, for example, to build a set of pre-defined aggregated trees based on long-term traffic measurement information.

In [8], this static problem is proved NP-complete and several algorithms have been proposed. These algorithms try to map this problem to Minimum Set Cover (MSC) problem. First, for each group, a native multicast tree is computed and then extended in a recursive

manner to enumerate all possible trees (called *candidate trees*) that cover the group without violating *bth*. After obtaining candidate trees for all groups, we can identify the groups that can be covered by each candidate tree. Finally, by solving MSC (using ILP or greedy algorithm), we can determine the minimum set of trees that cover all groups. The basic idea of the greedy algorithm is to iteratively find the candidate tree that covers the largest number of uncovered groups.

**Dynamic On-Line Trees** The dynamic version of the group-tree matching problem is useful for on-line systems when 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 assign a group to a tree when the group starts, with the same objective of minimizing the number of aggregated trees without violating *bth*. Several previous studies [11, 5] proposed heuristic dynamic algorithms, but there is neither comparison with optimal solutions nor comparison between themselves. In this paper, we mainly investigate the performance of these dynamic on-line algorithms.

### 3 A Generic Dynamic On-Line Algorithm and Analysis

A dynamic on-line group-tree matching algorithm needs to accommodate group dynamics, with the goal of maximizing the aggregation degree under a bandwidth waste constraint. 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 **GDOA**) which extends the various heuristics in the literature, and provide an upper-bound analysis on its performance.

#### 3.1 Algorithm Description

Considering group dynamics, we present the algorithm in two procedures: group join and group leave. As shown in Algorithm 1, 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  $g$ , it is extended to  $t^e$  using a greedy algorithm similar to Prim's minimum spanning tree algorithm [9] to connect  $t$  to uncovered nodes in  $g$ . 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 if  $t^e$  satisfies the bandwidth waste requirement for  $G^c(t) \cup g$ . If this is the case, then  $t^e$  is considered a candidate tree. Finally,

---

#### Algorithm 1 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:    extend tree  $t$  to  $t^e$  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:       $T^c(g) \leftarrow T^c(g) \cup t^e$ 
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:    $t^f(g) \leftarrow t$  with min.  $\delta(t, g)$ 
21: end if

```

---

if there is one or more candidate tree 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 group  $g$  must be adopted as the final tree.

When a group  $g$  leaves the network, Algorithm 2 will be activated. In order to obtain a better set of trees, tree  $t(g)$  will be shrunk whenever possible (line 5).

---

#### Algorithm 2 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  $t(g)$ 

```

---

**Complexity Analysis** Assume the number of nodes in the network is  $n$ , the number of links  $m$ , the number of existing groups  $N_g$ , and the number of trees  $N_t$ . In the above GDOA algorithm, the existing trees  $T$  are searched to find a best match for each group. Checking the eligibility of an existing tree or an extended tree and finding the native tree require  $O(n)$  time, so the time complexity for group join is  $O(N_t n) = O(N_g n)$  ( $N_t$  is bounded by  $N_g$ ). A similar analysis applies to group leave, which has the same time complexity of  $O(N_g n)$ .

### 3.2 An Upper Bound

For the dynamic on-line group-tree matching problem, it is very difficult to have a global optimal solution due to group dynamics. On the other hand, if we could obtain an optimal solution at any given time point, we can directly obtain an upper bound on the performance of the dynamic on-line algorithms. Thus, at every sampling point, we collect the set of currently active groups, run an optimal (or near-optimal) Static Pre-Defined Tree algorithm [8], 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 GDOA in terms of Aggregation Degree, since GDOA only considers the existing trees as candidates, while the optimal solution takes all possible trees into account. In Section 4, we will investigate the performance of GDOA by comparing it with the Greedy algorithm for Static Pre-Defined Tree (referred to as **UBAA-Greedy** in this paper). As shown in [8], the Greedy algorithm provides a near-optimal solution with a much faster computation speed compared with the optimal ILP algorithm.

**Complexity Analysis** From [8], the complexity of the Greedy algorithm is  $O(N_g^2 n^5)$  when bandwidth waste threshold is small. Obviously, UBAA-Greedy is much more time consuming than GDOA, which makes UBAA-Greedy impractical for on-line systems. In this paper, we mainly use UBAA-Greedy to evaluate the performance of dynamic on-line algorithms.

### 3.3 Discussion of Existing Dynamic Heuristics

In the literature, several dynamic on-line group-tree matching heuristics have been proposed [11, 5]. In [11], Song et. al. developed a protocol called **MTBF** (Multicast Tunnelling with Branching Forwarding) for the inter-domain multicast architecture MASC/BGMP. For a domain with  $n_e$  edge routers, only  $n_e$  aggregated trees are established, each rooted at one edge router. Groups with the same source edge router will share one tree. 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. In [5], ASSM is presented for backbone domain multicast provisioning. It essentially uses a simplified version of GDOA without tree extension and shrinking in order to reduce overhead. In this paper, we refer to this simple version of GDOA as **Simple-GDOA**. In next section, we will quantitatively evaluate the performance of algorithms of MTBF and Simple-GDOA.

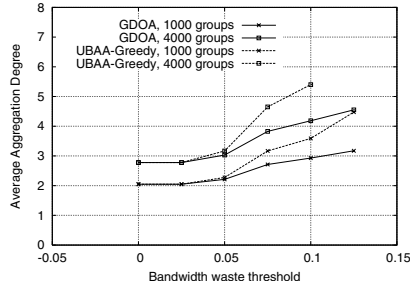


Figure 2. AD of GDOA and UBAA-Greedy.

## 4 Performance Evaluation

In this section, we first introduce the simulation environments and performance metrics. Then we present the simulation results of GDOA, Simple-GDOA, and MTBF, respectively.

In our simulations, we use a network topology abstracted from a real network topology AT&T IP backbone. In the abstracted topology, there are 9 gateway routers, 9 backbone routers, 18 access routers and 18 exchange nodes. Therefore, there are 18 core (gateway and backbone) routers and 36 edge (access and exchange) routers. We use the *Random Node Weight Model* [6] for group member generation. The multicast group requests arrive as a Poisson process and groups’s lifetime follows an exponential distribution. We vary the average group arrival rate to control the number of co-existing groups. After the steady state is reached, we start to collect simulation data at certain intervals. Simulation details can be found in [8].

We define the following two metrics besides Aggregation Degree (AD). 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 **State Reduction Ratio (SRR)** as the percentage of the multicast state entries in core routers reduced by tree aggregation. The higher the SRR, the more multicast state entries are reduced. **Tree Control Overhead (TCO)** is defined as the number of control messages for tree setup and removal, tree extension and shrinking in GDOA, and filter setup and removal in MTBF. It measures the control overhead of group-tree matching algorithms.

### 4.1 How Good is GDOA?

Recall that UBAA-Greedy gives an upper bound of GDOA performance. In our first set of experiments, we compare the performance of GDOA with UBAA-Greedy. We vary the bandwidth waste threshold  $bth$

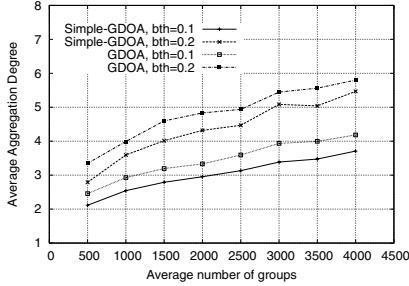


Figure 3. AD of GDOA and Simple-GDOA.

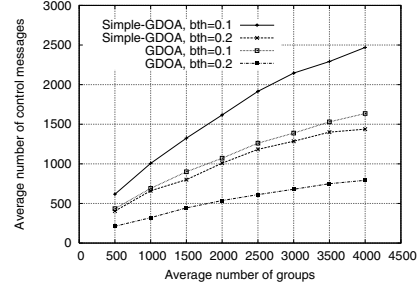


Figure 4. TCO of GDOA and Simple-GDOA.

from 0 to 0.125 (we tend to make  $bth$  small since UBAA-Greedy takes very long time when  $bth$  is big [8]).

As illustrated in Figure 2, UBAA-Greedy bounds GDOA in all cases. For small  $bth$ , GDOA’s performance is very close to UBAA-Greedy. In addition, as  $bth$  increases, the difference between these two algorithms remains fairly stable for  $bth \geq 0.05$ . These results show that GDOA can achieve its goal reasonably well. In addition, AD increases when  $bth$  is lifted. This is exactly the trade-off which aggregated multicast seeks: the more bandwidth waste, the more aggregation can be achieved. Furthermore, when the number of groups is raised, the possibility of group overlapping increases, thus AD also goes up. In other words, aggregated multicast is scalable to the number of groups. This is also a general trend that can be observed in some other figures. For SRR, we observe a similar trend, and the figure is omitted to save space.

We also compared the execution time of GDOA and UBAA-Greedy (not shown here due to space limit). UBAA-Greedy generally takes longer time, especially when  $bth$  increases, due to the exhaustive search of candidate trees required by the algorithm.

#### 4.2 GDOA vs. Simple-GDOA

Simple-GDOA is a simple version of GDOA that does not involve tree extension and shrinking. To investigate the trade-off between GDOA and Simple-GDOA, we conduct a second set of experiments to compare these two algorithms.

In Figure 3, GDOA achieves better AD than Simple-GDOA. For example, GDOA has around 15% higher AD than Simple-GDOA when  $bth = 0.1$  and there are 1000 groups. This is due to the fact that tree extension and shrinking features in GDOA help to find a better tree set by “slightly changing” the trees. The figure of SRR (not given here) is also consistent with AD.

In Figure 4, the tree control overhead of Simple-

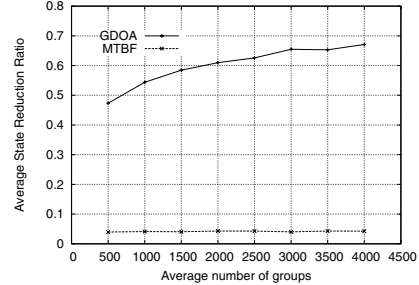


Figure 5. SRR of GDOA and MTBF.

GDOA is higher than that of GDOA, especially when the number of groups is big, which is counterintuitive. The explanation is that, in GDOA, even though tree extension and shrinking cause additional messages, the aggregation gain (i.e., less tree setup and removal overhead) compensates more than the expected loss. This shows that GDOA is actually a winner when there are many groups in the network.

From Figure 3 and 4, we also have the following observations. First, when  $bth$  increases, Simple-GDOA resembles GDOA to a higher extent for both AD and TCO, because it is easier to find candidate trees for a group and tree extension becomes less critical. Second, when there are more groups, the gap of the TCO between these two algorithms widens, which shows that GDOA’s aggregation gain has stronger effect on control overhead than the cost of tree extension and shrinking.

To summarize, Simple-GDOA is more suitable for the scenarios with bigger  $bth$  and smaller numbers of groups; on the other hand, GDOA performs better in all the cases and demonstrates its promise especially when  $bth$  is small and there are a large number of groups, which are the preferable scenarios in reality.

#### 4.3 GDOA vs. MTBF

To compare the performance of GDOA and MTBF, we plot the their SRR and TCO in Figure 5 and 6,

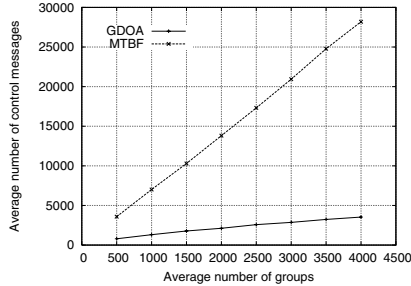


Figure 6. TCO of GDOA and MTBF.

when there is no bandwidth waste. Both figures tell us that GDOA significantly outperforms MTBF especially when there are a large number of groups. The performance of MTBF depends heavily on group member distribution: if a group is small compared with the aggregated multicast tree, then a lot of filters need to be installed in the routers in order to avoid bandwidth waste. In other words, a very small number of aggregated trees (as in MTBF) may not be flexible enough to accommodate a large number of multicast groups without significant performance penalty. GDOA, in contrast, is more flexible since it forces groups with similar membership to share an aggregated tree.

## 5 Conclusions

In this paper, we have formally studied the dynamic on-line group-tree matching problem in large scale group communications. We propose a generic dynamic on-line algorithm (GDOA) and provide an approach to determine the upper bound on its performance by utilizing some optimal (or near optimal) static group-tree matching solutions. We quantitatively evaluate the performance of GDOA and some existing dynamic on-line heuristics (Simple-GDOA and MTBF). Our experiment results can be summarized as follows: 1) GDOA performs reasonably close to its upper bound with much faster computation time; 2) Compared with GDOA, Simple-GDOA is more suitable when  $bth$  is big and there are small number of groups, however, these scenarios are usually not common in the real systems; 3) MTBF sacrifices state reduction and introduces more control overhead when fixing the number of aggregated trees.

**Future Work** In this study, we seek to minimize the total number of aggregated trees given a bandwidth waste threshold. In the future work, it may be beneficial to minimize the wasted bandwidth when the maximum number of trees to be managed is fixed. For the purpose of comprehensive analysis, we also want to test

the algorithms in more topologies. Finally, we do not consider in this paper some bandwidth issues (such as link capacity limitation, background traffic, and load balance), which deserve further investigation.

## References

- [1] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, and O. Paridaens. Explicit multicast (Xcast) basic specification. *Internet draft*, Mar. 2001.
- [2] F. Y. Y. Cheng and R. 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] L. H. M. Costa, S. Fdida, and O. C. M. Duarte. Hop-by-hop multicast routing protocol. *Proceedings of SIGCOMM'01*, Aug. 2001.
- [5] 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.
- [6] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast with inter-group tree sharing. *Proceedings of NGC2001*, Nov. 2001.
- [7] P. Francis. Yoid: extending the internet multicast architecture. <http://www.aciri.org/yoid/docs/index.html>.
- [8] L. Lao, J.-H. Cui, and M. Gerla. Tackling group-tree matching problem in large scale group communications. Technical report, UCLA CSD TR040022, <http://www.cs.ucla.edu/NRL/hpi/AggMC/index.html>, June 2004.
- [9] U. Manber. *Introduction to Algorithms: a Creative Approach*. Addison-Wesley Publishing Company, 1989.
- [10] 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.
- [11] 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*, 2000.
- [12] 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.
- [13] D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. *Proceedings of IEEE INFOCOM*, Mar. 2000.
- [14] J. Tian and G. Neufeld. Forwarding state reduction for sparse mode multicast communications. *Proceedings of IEEE INFOCOM*, Mar. 1998.
- [15] D.-N. Yang and W. Liao. Optimizing State Allocation for Multicast Communications. In *Proceedings of IEEE INFOCOM*, Mar. 2004.