Topology-Aware Overlay Networks for Group Communication

Minseok Kwon
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398
kwonm@cs.purdue.edu

Sonia Fahmy
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398
fahmy@cs.purdue.edu

ABSTRACT

We propose an application level multicast approach, Topology Aware Grouping (TAG), which exploits underlying network topology information to build efficient overlay networks among multicast group members. TAG uses information about path overlap among members to construct a tree that reduces the overlay relative delay penalty, and reduces the number of duplicate copies of a packet on the same link. We study the properties of TAG, and model and experiment with its economies of scale factor to quantify its benefits compared to unicast and IP multicast. We also compare the TAG approach with the ESM approach in a variety of simulation configurations including a number of real Internet topologies and generated topologies. Our results indicate the effectiveness of the algorithm in reducing delays and duplicate packets, with reasonable algorithm time and space complexities.

Categories and Subject Descriptors

C.2.2 [**Network protocols**]: Applications; C.2.5 [**Local and wide-area networks**]: Internet; D.4.4 [**Communications management**]: Network communication; D.4.8 [**Performance**]: Simulation

General Terms

Algorithms, Design, Performance

Keywords

overlay networks, application level multicast, network topology, routing

1. INTRODUCTION

A variety of issues, both technical and commercial, have hampered the widespread deployment of IP multicast in the global Internet [8, 9]. Application level multicast approaches using overlay networks [4, 5, 6, 11, 15, 22, 30] have been recently proposed as a viable alternative to IP multicast. In particular, End System Multicast (ESM) [5, 6] has gained considerable attention due to its success in conferencing applications. The main idea of ESM (and its Narada protocol) is that end hosts exclusively handle group management, routing information exchange, and overlay forwarding tree construction. The efficiency of constructed large overlay multicast trees, in terms of both performance and scalability, is the primary subject of this paper.

We propose a simple algorithm, which we call Topology Aware Grouping (TAG), to exploit underlying network topology data in constructing efficient overlays for application level multicast. Each new member of a multicast session determines the path from the root of the session to itself, and uses path overlap information to partially traverse the overlay data delivery tree and determine its parent and children. The path computed under current Internet routing protocols serves as the basis for building the overlay network with a small delay penalty and few duplicate packets sent on the same link due to overlaying. TAG nodes maintain a small amount of information– IP addresses and paths of only their parent and children nodes.

Unlike ESM, TAG is tailored to applications with a large number of members, which join the session at different times, and regard delay as a primary performance metric and bandwidth as a secondary metric. For example, in a limited bandwidth streaming application or multi-player on-line game, latency is an important performance measure. TAG constructs its overlay tree based on delay (as used by current Internet routing protocols), but uses bandwidth as a loose constraint, and to break ties among paths with similar delays. We investigate the properties of TAG and model its economies of scale factor, compared to unicast and IP multicast. We verify the economies of scale factor by simulation. We also demonstrate via simulations the effectiveness of TAG in terms of delay, duplicate packets, and available bandwidth in a number of large-scale configurations.

The remainder of this paper is organized as follows. Section 2 describes the basic algorithm and its extensions. Section 3 analyzes the properties of TAG and its economies of scale factor. Section 4 simulates the proposed algorithm and compares it to ESM using real and generated Internet topologies. Section 5 discusses related work. Finally, section 6 summarizes our conclusions and discusses future work.

2. TOPOLOGY AWARE OVERLAYS

Although overlays have emerged as a practical alternative to IP multicast, overlay performance in terms of delay penalty and number of duplicate packets (referred to as "link stress") in large groups have been important concerns. Moreover, exchange of overlay end-to-end routing and group management information limits the scalability of the overlay multicast approach. Most current overlay

NOSSDAV'02, May 12-14, 2002, Miami, Florida, USA.

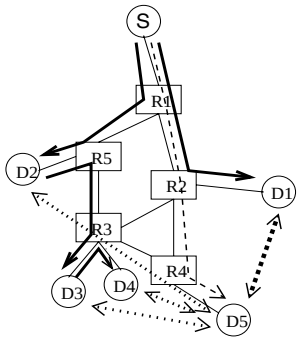

Figure 1: Example of topology aware overlay networks

network proposals employ two basic mechanisms: (1) a protocol for collecting end-to-end measurements among members; and (2) a protocol for building an overlay graph or tree using these measurements.

We propose to exploit the underlying network topology information for building efficient overlay networks. By "underlying network topology," we mean the shortest path information IP routers maintain. The definition of "shortest" depends on the particular routing protocol employed, but usually denotes shortest in terms of delay or number of hops. Using this topology information is illustrated in figure 1. In the figure, source S and destinations $D1$ to $D4$ are end hosts that belong to the multicast group, while $R1$ to $R5$ represent routers. Thick solid lines denote the current data delivery tree from S to $D1 - D4$. The dashed line represents the shortest path to a new node $D5$ from S. If $D5$ wishes to join the delivery tree, which member is the most appropriate parent node to $D5$? A relay from $D1$ to $D5$ is consistent with the shortest path from S to $D5$, and no duplicate packet copies would be added (packets in one direction are counted separately from packets in the reverse direction). It is difficult to determine the shortest path and the number of duplicate packets introduced, in the absence of knowledge of the underlying network topology. If network topology information can be obtained by the multicast participant (as discussed in section 2.4), nodes need not exchange complete end-to-end measurements, and topology information can be exploited to construct better overlay trees. A TAG destination selects as a parent the destination whose shortest path from the source has maximal overlap with its own path, to reduce increase in number of hops (and hence reduce delay) over a unicast path, while satisfying loose bandwidth constraints.

Like all overlay approaches, TAG does not require class D addressing or multicast router support. A TAG session can be identified by (root_IP_addr, root_port), where "root" denotes the the primary sender in a session, which serves as the root of the multicast delivery tree. The case of multiple senders will be discussed in section 2.5. We define the following terms, which will be used throughout the paper:

DEFINITION 1. *A $path$ from node A to node B, denoted by $P(A, B)$, is a sequence of routers comprising the shortest path from node A to node B according to the underlying routing protocol. $P(S, A)$ will be referred to as the $spath$ of A where S is the root of the tree. The $length$ of a path P or $len(P)$ is the number of routers in the path.*

DEFINITION 2. *$A \succ B$ if $P(S, A)$ is a prefix of $P(S, B)$ where S is the root of the tree.*

For example, the $path$ from S to $D5$ (or $spath$ of $D5$) in figure 1 is $P(S, D5) =< R1, R2, R4 >$ with $len(P(S, D5)) = 3$. $D1 \succ D5$.

A TAG node maintains a family table (FT) defining parent-child relationships for this node. One FT entry is designated for the parent node and the remaining entries are for the children. As seen in figure 2, an FT entry consists of a tuple ($address$, $spath$). The $address$ is the IP address of the node and the $spath$ is the shortest path from the root to this node.

2.1 Complete Path Matching Algorithm

The path matching algorithm traverses the overlay data delivery tree to determine the best parent (and possibly children) for a new node. The algorithm considers three mutually exclusive conditions as depicted in figure 3. Let N be a new member wishing to join a current session and C be the node being examined. If possible, we select a node A such that A is a child of C, $A \succ N$, and $len(P(S, N)) > len(P(S, A)) > len(P(S, C))$, and continue traversing the sub-tree rooted at A (figure 3(a)). Otherwise, if there are children A_i of C such that $N \succ A_i$ for some i, N becomes a child of C with A_i as N's children (figure 3(b)). In case no child of C satisfying the first or second conditions exists, N becomes C's child (figure 3(c)). Note that no more than one child satisfying the first condition can exist. The complete path matching algorithm is presented in figure 4. In the algorithm, N denotes a new member; C represents the node currently being examined by N; and $target$ is the next node which N will probe, if necessary.

```
proc PathMatch(C, N) ≡
  ch := first child of C;
  flag := condition(3);
  while (ch is NOT NULL) do
      if (ch ≻ N)
        then
            target := ch;
            flag := condition(1); fi;
      if (N ≻ ch)
        then
            add ch to children(N);
            N becomes parent(ch);
            flag := condition(2); fi;
      if (flag is NOT condition(1))
        then
            ch := next child of C;
         else
            ch := NULL;
      fi;
  od;
  if (flag is condition(1))
    then
        PathMatch(target, N);
     else
        add N to children(C);
  fi;
  Variables :
    C : node currently being examined
    N : new node joining the group
    ch : a child of C
    target : next node N will examine
```

Figure 4: Complete path matching algorithm

There are two reasons for selecting a node (in the first and second conditions) whose $spath$ is the longest prefix of the $spath$ of

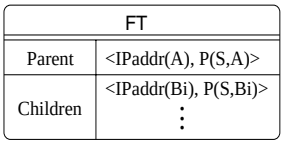

Figure 2: Family table (FT)

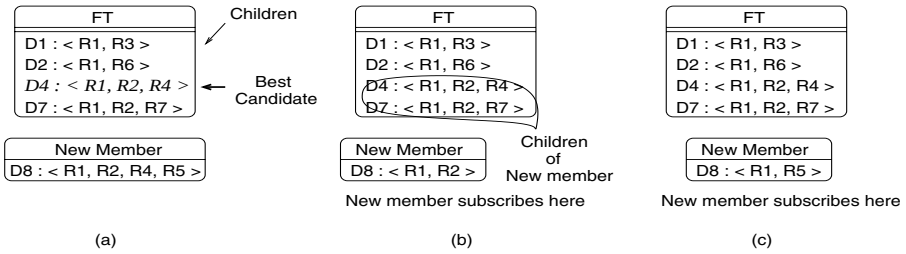

Figure 3: The three conditions for complete path matching

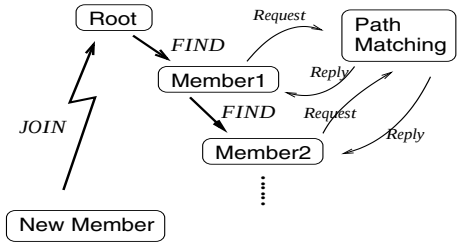

Figure 5: A member join process in TAG

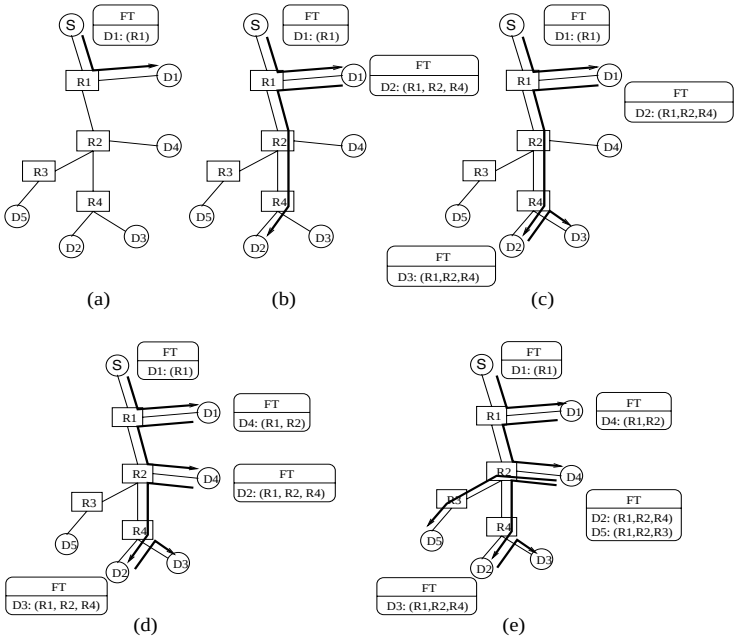

Figure 6: Member join in TAG

the new member. First, the path from the source to the new member will be consistent with the shortest path determined by routing algorithms. This reduces additional delay introduced by overlays. Second, sharing the longest prefix curtails the number of duplicate packets in the overlay network, since the additional hops from the parent to the child are reduced.

2.2 Tree Management

In this section, we discuss the multicast tree management protocol including member join, member leave, fault resilience, and adaptivity, and evaluate its time complexity.

2.2.1 Member Join

A new member joining a session sends a JOIN message to the main sender S of the session (the root of the tree). Upon the receipt of a JOIN, S computes the $spath$ to the new member, and executes the path matching algorithm. If the new member becomes a child of S, the FT of S is updated accordingly. Otherwise, S propagates a FIND message to its child that shares the longest $spath$ prefix with the new member $spath$. The FIND message carries the IP address and the $spath$ of the new member, and is processed by executing path matching and either updating the FT, or propagating the FIND. The propagation of FIND messages continues until the new member finds a parent. The process is depicted in figure 5.

An example is illustrated in figure 6. Source S is the root of the multicast tree; $R1$ through $R4$ are routers, and $D1$ through $D5$ are destinations. The thick arrows denote the multicast forwarding tree computed by TAG. The FT of each member is shown next to it. The destinations join the session in the order $D1$ to $D5$. Upon the receipt of a JOIN message from $D1$, S creates an entry for $D1$ in its FT (figure 6(a)). S computes the shortest path for a destination upon receiving the JOIN message of that destination. When $D2$ joins the session (figure 6(b)), S executes the path matching algorithm with the $spath$ of $D2$. S determines that $D1$ is a better parent for $D2$ than itself, and sends a FIND message to $D1$ which takes $D2$ as its child. $D3$ similarly determines $D2$ to be its parent (figure 6(c)). When $D4$ joins the session, $D4$ determines $D1$ to be its parent and takes $D2$ and $D3$ as its children. The FTs of $D1$ and $D4$ are updated accordingly (figure 6(d)). Finally, $D5$ joins the session as a child of $D4$ (figure 6(e)). Figure 6(e) gives the final state of the multicast forwarding tree and the FT at each node.

2.2.2 Member Leave

A member can leave the session by sending a LEAVE message to its parent. For example, if $D4$ wishes to leave the session (figure 7(a)), $D4$ sends a LEAVE message to its parent $D1$. A LEAVE message includes the FT of the leaving member. Upon receiving LEAVE from $D4$, $D1$ removes $D4$ from its FT and adds FT entries for the children of $D4$ ($D2$ and $D5$ in this case). The constructed multicast forwarding tree is illustrated in figure 7(b).

2.2.3 Time Complexity

Assume, for simplicity, that a tree node has an average of k children, and assume n total multicast participants. After the source discovers the path to the new member, the member join process entails: (1) operations within each node and (2) tree traversal. Operation (1) is of order $k(\log_k n)$ since k is the number of children to be examined and $\log_k n$ is time for the longest path matching. Operation (2), in the worst case, requires $\log_k n$ operations from the root to a leaf node. Therefore, the total time complexity of member join is order $k(\log_k n)^2$. Member leave requires processing k FT entries with $\log_k n$ worst case path length. The total complexity of

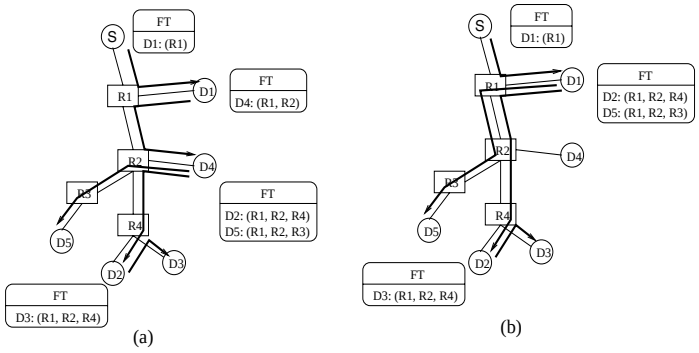

Figure 7: Member leave in TAG

member leave is order $k(\log_k n)$.

2.2.4 Fault Resilience

Failures in end hosts of a multicast session (not an uncommon occurrence) affect all members of the subtree rooted at the failing node. To detect failures, a parent and its children periodically exchange reachability messages in the absence of data. When a child failure is detected, the parent simply discards the child from its FT, but when a parent failure is detected, the child must rejoin the session.

2.2.5 Adaptivity to Changes

If network conditions change and the overlay tree becomes inefficient (delays become large, e.g., when a mobile host moves or paths fail), TAG must adapt the overlay tree to the new network conditions. An accurate adaptation would entail that the root probe every destination periodically to determine if the paths have changed. When path changes are detected, the root initiates a rejoin process for the destinations affected. This mechanism, however, introduces scalability problems in that the root is over-burdened and many potential probe packets are generated.

We propose three mechanisms to mitigate these scalability problems. First, intermediate nodes (non-leaf nodes) participate in periodic probing, alleviating the burden on the root. The intermediate nodes only probe the paths to their children. Second, path-based aggregation of destinations can substantially reduce the number of hops and destinations probed. Destinations are aggregated if they have the same $spath$. Only one destination in a destination group is examined every round. During the next round, another member of the group is inspected. When changes are detected for a certain group, all members in that group are updated. Third, when changes in part of the $spath$ are detected for a destination, not only the destination being probed, but all the destinations in the same group and all the destinations in groups with overlapping $spath$s are updated.

2.3 Bandwidth Considerations

Since TAG targets delay-sensitive applications, delay (as used by the underlying IP routing protocol) is the primary factor used in path matching. The complete path matching algorithm presented in figure 4 reduces the delay from source to destination, and reduces the total number of duplicate packets. However, high link stress (and limited bandwidth to each child) may be experienced near a few high degree or limited bandwidth nodes in the constructed tree. To alleviate this problem, we loosen the path matching rule when a node is searching for a parent. The new rule allows a node B to attach to a node A as a child if A has a common $spath$ prefix of length $len(P(S, A)) - k$ with B, even if the remaining k elements of the $spath$ of A do not match the $spath$ of B. We call this method minus-k (or partial) path matching.

Minus-k partial path matching allows *children* of a bandwidth-constrained node to take on new nodes as their children, mitigating the high stress and limited bandwidth near the constrained node. When the available bandwidth at a given node falls below a threshold $bwthresh$, minus-k path matching is activated. The threshold $bwthresh$ does not denote a strict guarantee, but it gives an indication that alternate paths should be explored. The k parameter controls the deviation allowed in the path matching. A large value of k may increase the delay, but it reduces the maximum stress a node may experience and improves bandwidth.

With minus-k matching, a new member can examine several delay-based paths while traversing the tree, and select the path which maximizes bandwidth. When a node C is being probed by the new member, all children of C eligible to be potential ancestors of the new member (by the minus-k path matching algorithm) constitute a set of nodes to examine next. The node which achieves the maximum bandwidth among these nodes is the one selected. The new path matching algorithm that takes bandwidth as a secondary metric is presented in figure 8.

```
proc PathMatch(C, N) ≡
  ch := first child of C;
  flag := condition(3);
  maxbw := 0;
  if (bandwidth(C) < bwthresh)
    then
        minus-k path matching in condition(1) activated;
  fi;
  while (ch is NOT NULL) do
        if (ch ≻ N && bandwidth(ch) > maxbw)
          then
                target := ch;
                maxbw := bandwidth(ch);
                flag := condition(1); fi;
        if (N ≻ ch)
          then
                add ch to children(N);
                N becomes parent(ch);
                flag := condition(2); fi;
        ch := next child of C;
  od;
  if (flag is condition(1))
    then
        PathMatch(target, N);
     else
        add N to children(C);
  fi;
  Variables :
    C : node currently being examined
    N : new node joining the group
    ch : a child of C
    target : next node N will examine
    maxbw : maximum bandwidth among potential
            parents of N
```

Figure 8: Partial path matching algorithm, with bandwidth as a secondary metric. The bandwidth() function gives the available bandwidth between a node and the new member

Table 1 shows the tradeoff between the delay (mean RDP as defined in section 4), total duplicate packets on the tree, and maximum link stress in the tree for a variety of $bwthresh$ values. The

simulation setup used will be discussed in section 4. The configuration used here is TAG-TS2 with 1000 members. We use a fixed $k = 1$ in these simulations, though we are currently investigating dynamic adaptation of k. As seen in the table, as $bwthresh$ increases, minus-k path matching is activated more often. Consequently, a larger $bwthresh$ value reduces the total number of duplicate packets and maximum stress, but increases the RDP value. If TAG does not use minus-k path matching ($bwthresh$=0), TAG trees suffer from a large number of duplicate packets and high maximum stress, yielding less bandwidth per child.

Table 1: Tradeoffs with different $bwthresh$ values

$bwthresh$	RDP	Duplicate packets	Max. stress
400	1.590270	1284	74
300	1.573488	1379	76
200	1.522141	1423	78
100	1.448861	1528	95
50	1.390421	1975	108
20	1.305699	1960	111
0	1.335015	3982	340

2.4 Topology and Bandwidth Data

When a new member joins a multicast session in TAG, the root must obtain the path from itself to the new member. We propose two possible approaches to perform this.

The first approach is to use a network path finding tool such as *traceroute*. *Traceroute* has been used as a tool for network topology discovery [12, 21]. Some routers, however, do not send ICMP messages when TTL reaches zero for several reasons, the most important of which is security. To investigate this, we conducted simple experiments where we used *traceroute* for 50 sites in different continents at different times. Approximately 90% of the routers responded, while around 10% did not respond. The average time taken to obtain and print the entire path information was 7.5 seconds, with a maximum of 26.3 seconds and a minimum of 0.6 seconds.

The second option is to exploit topology information provided by topology servers. For example, an OSPF topology server [26] can track intra-domain topology, either by listening to OSPF link state advertisements, or by pushing and pulling information from routers via SNMP. Internet topology discovery projects, e.g., [12, 21] can supply inter-domain topology information to TAG when a new member joins or changes occur. Topology servers may, however, only include partial or coarse-grained (e.g., AS-level) information. Partial information can still be exploited by TAG for partial path matching of longest common subsequences.

Instability in dynamic network conditions and bandwidth estimation tools may also be useful for TAG, in conjunction with in-band measurements. Tools similar to *pathchar* [13] are useful for estimating the available bandwidth, delay, average queue, and loss rate of every hop between any source and destination on the Internet. *Nettimer* [17] is useful for low-overhead measurement of per-link available bandwidth.

2.5 Multiple Sender Groups

In the current version of TAG, a sender other than the root of the tree must first relay its data to the root. The root then multicasts the data to all members of the multicast group. This approach is suitable for mostly single-sender applications, where the main sender is selected as the root, and other group members may occasionally transmit. In applications where all members transmit with approximately equal probabilities, the root of the tree should be carefully selected, as with core selection in core-based tree approaches for multicast routing [3]. Multiple (backup) roots are also important for fault tolerance.

3. ANALYSIS OF TAG

In this section, we investigate various properties of TAG and study its economic aspect. For simplicity, we use TAG with complete path matching (figure 4) in our analysis, without incorporating the bandwidth constraints and optimizations. However, the results can be easily extended to minus-k path matching.

3.1 Properties of TAG

We study the three conditions in the path matching algorithm, and several properties of the trees constructed by TAG.

LEMMA 1. *Node A is an $ancestor$ of node B in the TAG tree iff $A \succ B$.*

$Proof$: $\Rightarrow$: We first show that if node X is the parent of node Y, denoted by $X = Parent(Y)$, then $X \succ Y$. In the path matching algorithm, X can become the parent of Y by the second or the third path matching conditions. Both cases guarantee $X \succ Y$.

Then, we generalize to the case when node A is an $ancestor$ (not necessarily the parents) of node B. In this case, there must be n ($n > 0$) nodes such that $M_1 = Parent(B)$, $M_2 = Parent(M_1)$, $\ldots$, $M_n = Parent(M_{n-1})$, $A = Parent(M_n)$. $M_1 = Parent(B) \succ B$ holds, according to the previous case. Similarly, $M_2 \succ M_1 \succ B$. Transitively, $A = Parent(M_n) \succ B$.

$\Leftarrow$: This follows from conditions 1 and 2 in the path matching algorithm. □

Considering figure 1 as an example, node $D1$ is an $ancestor$ of node $D5$ because $P(S, D1) = < R1, R2 >$ is a prefix of $P(S, D5) = < R1, R2, R4 >$. In contrast, the fact that $P(S, D2) = < R1, R5 >$ is not a prefix of $P(S, D5) = < R1, R2, R4 >$ implies that node $D2$ is not an ancestor of $D5$. We now investigate the conditions of the path matching algorithm.

LEMMA 2. *The three conditions in the TAG path matching algorithm (given in figure 3) are mutually exclusive (no two of the three conditions can occur simultaneously) and complete (no other case exists).*

$Proof$: First, we prove the mutual exclusion. To show the mutual exclusiveness is equivalent to proving no two conditions can hold simultaneously. The first and the third conditions, and the second and the third conditions cannot co-exist by definition. Therefore, we need to show that the first and second conditions cannot co-exist. Suppose the first and the second conditions occur simultaneously for a node C that is being examined. A new member N selects B, a child of C, such that $B \succ N$ for further probing by the first condition. By the second condition, there must exist a node B', another child of C, such that $N \succ B'$, and B and B' are siblings. However, in this case, the path matching algorithm would have ensured that B' would be a child of B, not a child of C, by lemma 1, since $B \succ N \succ B'$. This is a contradiction.

Since the third condition includes the complement set of the first and the second conditions, the conditions are complete. □

Now we study the number of trees TAG can construct.

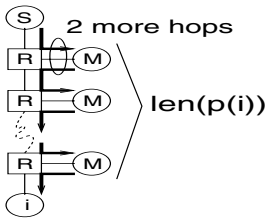

Figure 9: Delay bound in TAG

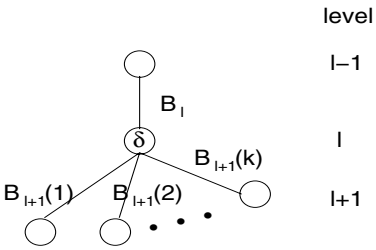

Figure 10: TAG Trees

LEMMA 3. *TAG constructs a unique tree if all members have distinct $spath$s, regardless of the order of joins. If there are at least two members with the same $spath$s, the order of joins alters the constructed tree.*

$Proof$: By lemma 1, a unique relationship among every two nodes (i.e., parent, ancestor, child, descendent, or none) is established among every two nodes which have different $spath$s, independent of the order of joins. If two members have the same $spath$, one must be an ancestor of the other. Therefore, $n!$ distinct trees can be constructed by TAG if n group members have the same $spath$s (according to the order of their joins). □

We study the parent properties next.

LEMMA 4. *For all i, the $spath$ of the parent of nodes A_i has the longest prefix of $spath$ $P(S, A_i)$, where "longest" denotes longest in comparison to the $spath$s of all members in a session.*

$Proof$: Consider two nodes B and C where B is the parent of C. By lemma 1, $B \succ C$, i.e., $P(S, B)$ is a prefix of $P(S, C)$. Suppose there exists a node A such that $P(S, A)$ is a prefix of $P(S, C)$ and $len(P(S, A)) > len(P(S, B))$. If both $P(S, A)$ and $P(S, B)$ are prefixes of the same path $P(S, C)$ and $len(P(S, A)) > len(P(S, B))$ means $P(S, B)$ is a prefix of $P(S, A)$. By definition 2, $B \succ A$, and therefore A must be a descendent of B by lemma 1. By the path matching algorithm, C ought to be a child of A instead of B since $B \succ A$ and $A \succ C$. This is a contradiction. Hence, $P(S, B)$ must be the longest prefix of $P(S, C)$, where B is the parent of C. □

Finally, we give a delay bound (in number of hops) on the path from a root to each member.

LEMMA 5. *For every destination i in a TAG tree, $SPD(i) \leq E(i) \leq 3 \times SPD(i) - 2$, where $SPD(i)$ is the shortest path delay from root S to i, and $E(i)$ is the actual delay from S to i in the TAG tree.*

$Proof$: Consider $P(S, i)$, the path from root S to i. $SPD(i) = len(P(S, i)) + 1$ since $SPD(i)$ is the number of hops in the path $P(S, i)$. The fact that $SPD(i)$ is the shortest path delay of $P(S, i)$ ensures $SPD(i) \leq E(i)$. Also, the maximum $E(i)$ occurs when i has as many ancestors as $len(P(S, i))$. This situation is depicted in figure 9. In the figure, R denotes a router; i is a destination, and nodes M are all ancestors of i. For every M, 2 more are added to the path. Thus, $2 \times len(P(S, i))$ more hops are added to $SPD(i)$. Therefore, the maximum $E(i)$ is $3 \times SPD(i) - 2$. □

3.2 Modeling the Economies of Scale Factor

Two important questions to answer about an overlay multicast tree are: (1) how much bandwidth it saves compared to unicast; and (2) how much additional bandwidth it consumes compared to IP multicast. IP multicast savings over naive unicast have been studied in [1, 7, 23]. In this section, we quantify the network load consumed by TAG. We derive a bound for the function $L_{TAG}(n)$ which denotes the total number of packet hops consumed by all n members of a multicast group.

For simplicity (as in [1, 23]), we assume a k-ary data propagation tree in which tree nodes denote routers. The height of the tree is H and all nodes except the leaves have degree k. An end host can be connected to any router (node in the tree). For simplicity, we assume that no unary nodes (nodes at which no branching occurs) exist in the tree. Therefore, the results are very approximate. Suppose that n end hosts join the multicast session. The probability that at least one end host is connected to a given node is:

$$p_n = 1 - (1 - \frac{1}{M})^n \tag{1}$$

where $M = 1 + k + k^2 + \cdots + k^H = \frac{k^{H+1}-1}{k-1}$ is the number of possible locations for an end host, which is equal to the number of nodes in the tree.

We now evaluate the number of packet hops (number of duplicate packets) consumed by the transmission at each level of the tree. In figure 10, B_l indicates the number of (packet) hops over the link between node δ at level l and its parent at level $l-1$. We compute B_l considering two different cases: when at least one host is connected to node δ, and when no host is connected to δ. Let B_l^1 be the number of hops in the first case while B_l^2 denotes the second case. According to TAG, the first case has only a single packet hop over the link, between node δ and its parent since node δ sends packets from the parent to the children ($B_l = 1$). In the second case, however, since no host relays the packets at δ, the total number of packet hops over outgoing links of δ towards the leaves is the number of packet hops over the link between δ and its parent. Therefore, $B_l^1 = p_n \cdot 1$ and $B_l^2 = (1 - p_n) \sum_{a=1}^{k} B_{l+1}(a)$. We assume the end hosts are uniformly distributed at nodes. This assumption implies that $E[B_{l+1}(1)] = E[B_{l+1}(2)] = \cdots = E[B_{l+1}(k)] = E[B_{l+1}]$. Therefore, $E[B_l^2] = (1 - p_n)kE[B_{l+1}]$. Hence, $E[B_l]$ is defined as follows:

$$E[B_l] = E[B_l^1] + E[B_l^2] = p_n + (1 - p_n)kE[B_{l+1}] \tag{2}$$

$$E[B_H] = 1 - (1 - p_n)^k \tag{3}$$

Additional packet hops are consumed by the link from a router to an end host connected on one of the router subnets. Observe that each end host uses 1 hop for only receiving data or 2 hops for receiving and relaying data. Thus, $n \leq A(n) \leq 2n$ holds where $A(n)$ denotes additional hop counts. Hence, the total number of packet hops $L_{TAG}(n)$ is:

$$\sum_{l=1}^{H} k^l E[B_l] + n \leq L_{TAG}(n) \leq \sum_{l=1}^{H} k^l E[B_l] + 2n \tag{4}$$

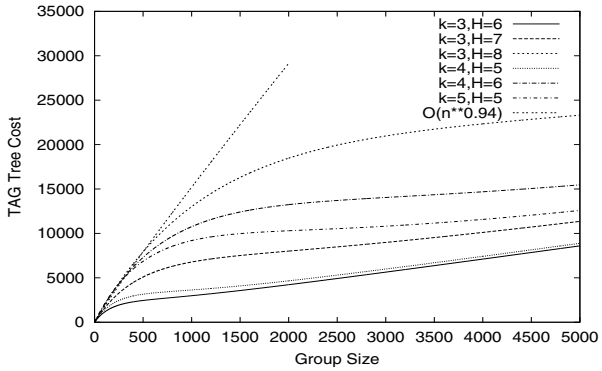

Figure 11: TAG tree cost versus group size

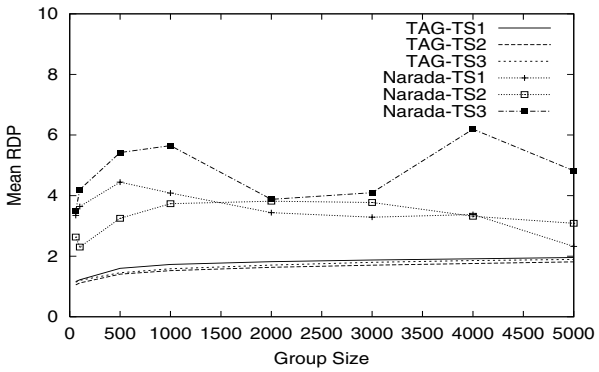

Figure 12: Mean RDP: TAG versus ESM

Solving the recurrence in (2) and (3), we obtain:

$$\sum_{l=1}^{H} k^l E[B_l] = \frac{k^H(1-(1-p_n)^k)(1-(1-p_n)^H)}{p_n} + \frac{k^H p_n}{(1-p_n)k-1}\left(\frac{1-(1-p_n)^H}{p_n} - \frac{k-k^{-H+1}}{k-1}\right) \quad (5)$$

Figure 11 plots the total cost of TAG for a variety of k and H values. All curves stabilize for group sizes exceeding 1500 members or so. With larger k and H values, the cost only stabilizes for a larger group. The slope decreases because as group size grows, more end hosts can share the links yielding more bandwidth savings. This is an important advantage of TAG over unicast. The figure shows that, very approximately, $L_{TAG}(n) \propto n^{0.94}$ before the curves stabilize. The factor 0.94 is smaller than with unicast, but larger than with IP multicast ($L_{IPmulticast}(n) \propto n^{0.8}$), where replication at the routers, together with good multicast routing algorithms yield additional savings. We will verify these results via simulations in section 4.3.

4. PERFORMANCE EVALUATION

We first discuss the simulation setup and metrics, and then analyze the results.

4.1 Simulation Setup and Metrics

We have implemented session-level (not packet-level) simulators for both TAG and ESM [6] to evaluate and compare their performance. Two sets of simulations were performed on different topologies. The first set uses Transit-Stub topologies generated by GT-ITM [29]. The Transit-Stub model represents router-level Internet topologies. We also simulate AS-level topologies, specifically the actual Internet AS topologies from NLANR [19] and topologies generated by Inet [16].

The Transit-Stub model produces a two-level hierarchy: interconnected higher level (transit) domains and lower level (stub) domains. We use three different topologies with different numbers of nodes: 492, 984, and 1640. We label them TS1, TS2, and TS3 respectively, e.g., label "TAG-TS1" denotes the results of TAG on the Transit-Stub topology with 492 nodes. Multicast group members are assigned to stub nodes randomly. The multicast group size ranges from 60 to 5000 nodes. GT-ITM generates symmetric link delays ranging from 1 to 55 ms. We use 1 to 3 ms for delays within a stub. We randomly assign bandwidths ranging between 100 MB and 500 MB to backbone links, and use the range 500 KB to 1 MB for the links from edge routers to end hosts.

The AS topologies from NLANR and Inet represent AS-level connectivity, not router-level connectivity. AS-level maps have been shown to exhibit a power-law topology [10], where a few nodes have high-degree connectivities while most other nodes exhibit low-degree connectivities. We use the 1997 NLANR data set, named AS97, and Inet data set for 1997 named Inet97, which has the same number of ASes as the NLANR data set: 3015. We generate 4000 members for each multicast session, and assign members to ASes randomly. Link delays and bandwidths in the same ranges as the Transit-Stub configuration are used for the AS configuration. The link delays are asymmetric.

We assume that the IP layer routing algorithm uses delay as a metric. The routing algorithm for the mesh in ESM uses available bandwidth as the primary metric and delay as a tie breaker. The minus-k path matching algorithm is used in TAG with a fixed $k = 1$ and $bwthresh = 200$ KB. We use the same parameters for ESM used in the simulations in [5, 6].

We use the following performance metrics [5, 6] for evaluating the TAG and ESM trees:

1. **Relative Delay Penalty (RDP):** The relative increase in delay between two nodes in TAG against unicast delay between the same two nodes;
2. **Link Stress:** Total number of duplicate copies of a packet over a physical link. We compute both the total stress for all tree links, and the maximum value of link stress among all links;
3. **Mean Available Bandwidth:** The mean available end-to-end bandwidth between every two nodes.

4.2 Performance Results

4.2.1 Transit-Stub Topologies

The mean RDP values produced by TAG and ESM for the three different Transit-Stub topologies (TS1, TS2, TS3) are plotted in figure 12. From the figure, TAG exhibits lower mean RDP values than ESM for different group sizes in all 3 topologies. The mean RDP values for TAG-TS1, TS2, TS3 are all in the range of 1 to 2. Although TS3 is a larger scale topology than TS2, and TS2 is larger than TS1, the mean RDP values for TAG are similar for the different topologies. Mean RDP values for TAG increase with the increase in group size. In ESM, mean RDP values range from 2 to 6. We observe that, contrary to TAG, the mean RDP values for ESM do not always increase with the increase in group size.

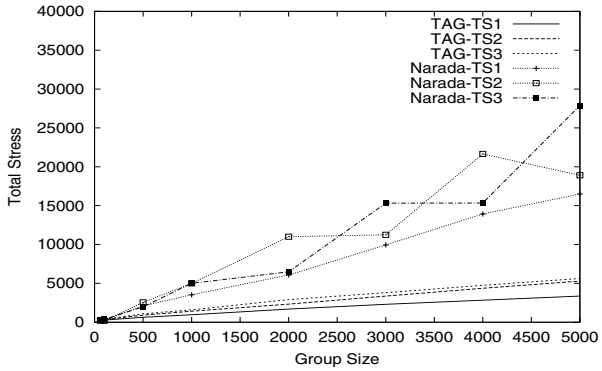

Figure 13: Total stress: TAG versus ESM

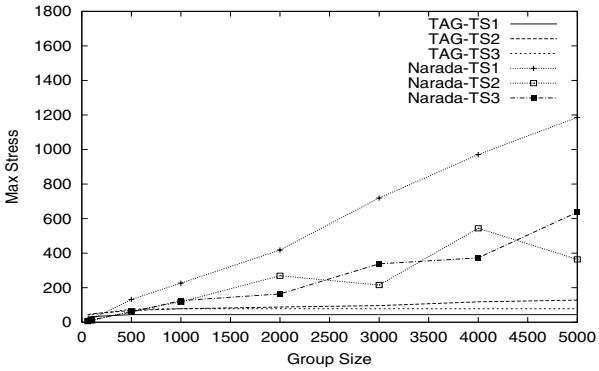

Figure 15: Maximum stress: TAG versus ESM

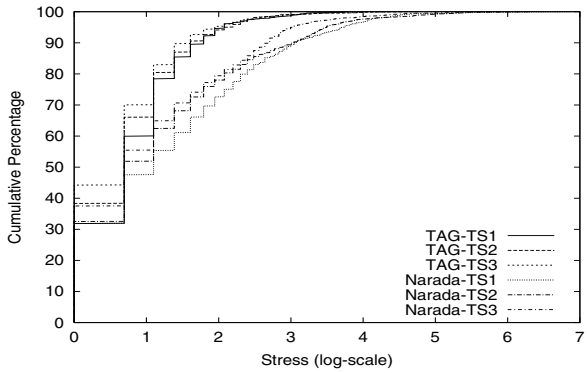

Figure 14: Cumulative distribution of stress

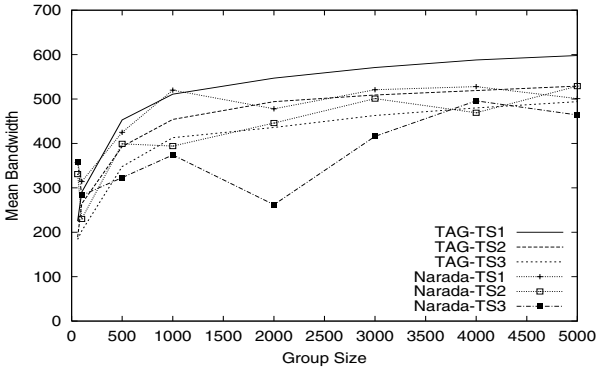

Figure 16: Mean BW: TAG versus ESM

Figure 13 illustrates the total stress of TAG and ESM for the three different topologies (TS1, TS2, TS3). For all group sizes and for all topologies, TAG stress is below 5000. In contrast, ESM exhibits higher stress. The total stress of all 6 configurations increases in proportion to the group size. TAG-TS1 exhibits the lowest total stress. Figure 14 depicts the cumulative distribution of total stress. The figure shows clearly that the three TAG configurations have a larger number of small stress links than the three ESM configurations. The three TAG configurations have similar distributions in approximately the same region. The three ESM configurations exhibit similar distributions as well. The reason for this is that delivery trees are constructed by TAG or ESM protocols in different manners with different metrics and optimization goals. Figure 15 illustrates that TAG reduces the maximum stress value as well. With the complete path matching algorithm, a strategically located end host attached to a high-degree router can be the parent of numerous nodes, which severely constrains the bandwidth available for each of these nodes, and increases the stress at this host. The minus-k partial path matching algorithm remedies this weakness, as shown in the figure.

The mean bandwidth plots depicted in figure 16 indicate the average available bandwidth to all members. The available bandwidth to a member is computed as the fair share bandwidth on the bottleneck link over the path to that member. No significant difference can be observed between TAG and ESM for all configurations.

An important point to note is that in the ESM algorithm, the lower and upper degree bounds on each group member are of crucial importance. The two parameters control the number of neighbors that each member communicates with. The upper bound makes the protocol scalable. In our simulations, we observe that increasing the upper degree bound for ESM significantly reduces the delay penalty and link stress. Clearly, having more neighbors enables a member to find a better path, resulting in lower delay penalty and lower link stress. The higher upper bound, however, increases the routing information exchanged, which is detrimental to scalability. For example, we simulated the topology Narada (ESM)-TS2 with a group size of 5000, but used a lower degree bound (LDB) of 3, and an upper degree bound (UDB) of 12, instead of a UDB value of 6 as used in the ESM simulations in [5, 6]. Instead of a mean RDP value of almost 3 (figure 12), mean RDP fell to 1.53 (lower than TAG). The total link stress fell to 6569 (higher than TAG) instead of almost 20,000 as shown in figure 13. The maximum stress was reduced from almost 400 (figure 15) to 138 (higher than TAG). The parameter choices for TAG, most significantly $bwthresh$ (which should be tuned according to application bandwidth requirements), also significantly affect the results. For example, setting $bwthresh$ to zero and using complete path matching dramatically improves the RDP values for TAG, at the expense of the maximum stress and bandwidth results, as discussed in section 2.3.

4.2.2 AS and Inet Topologies

Table 2 shows the mean RDP values of TAG and ESM trees on AS97 and Inet97. In terms of the mean RDP on AS97, ESM performs better than TAG while, on the Inet97 configuration, TAG is better than ESM. No significant difference can be observed between TAG and ESM with respect to the total link stress and the maximum stress. The mean bandwidth, however, is higher for ESM. These results are due to the fact that fanout of nodes in AS-

Table 2: Performance of TAG and ESM in AS and Inet

Configuration	Mean RDP	Total link stress	Max. stress	Mean bandwidth
TAG-AS97	4.691708	12162	291	172.00
Narada (ESM)-AS97	3.474963	13665	411	408.00
TAG-Inet97	4.875883	12893	404	167.00
Narada (ESM)-Inet97	6.242306	11103	310	371.00

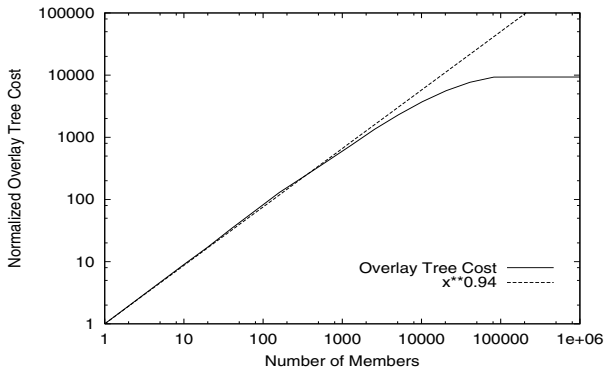

Figure 17: Overlay tree cost versus group size

level topologies is higher than that in router-level topologies. The minus-k path matching algorithm in TAG increases the RDP for the nodes which can no longer take a high-degree node as a parent. The available bandwidth at the high-degree node is reduced by the possibly large number of children. Since TAG only considers bandwidth as a secondary metric, it does not maximize bandwidth in its tree construction choices. As previously discussed, running the same simulations for TAG with a $bwthresh$ of zero dramatically reduces the RDP (to 1.5 for AS97 and 1.6 for Inet97) and the total stress (to less than 4000 for AS97 and around 5000 for Inet97), at the expense of an increase in the maximum stress and significant decrease in bandwidth.

4.3 Economies of Scale Factor

We simulate overlay tree bandwidth consumption to verify our results from section 3.2. In order to compare results, we assume that one hop used by one point-to-point transfer represents a unit of bandwidth, and count the total number of hops consumed by all members. We run three sets of simulations for unicast, TAG and IP multicast on the AS97 configuration. The complete (not minus-k) path matching TAG is used for a fair comparison of simulation results with the analysis results (which modeled complete path matching). The simulations show that unicast, TAG, and IP multicast consume 18820, 6553, and 3759 hops respectively. We also plot the normalized overlay tree cost of TAG for a variety of group sizes (using the same methodology as in [7]) in figure 17 on a log-log scale. The normalized overlay tree cost $L_{TAG}(m)/\hat{u}$ is defined as $L_{TAG}(m)$, the total number of hops consumed by all members m, divided by the average number of hops for unicast, $\hat{u}$. The figure shows that $L_{TAG}(m)/\hat{u} \propto m^{0.94}$. The overlay tree cost also stabilizes with tree saturation (as with IP multicast). This is consistent with our modeling results.

5. RELATED WORK

End system multicast (or Narada) [5, 6] is a clever architecture targeting small, sparse groups such as audio and video conferencing groups. End hosts in ESM exchange group membership information and routing information; build a mesh; and finally run a DVMRP-like protocol to construct a multicast forwarding tree. The authors show that it is important to consider both bandwidth (primarily) and latency when constructing conferencing overlays. However, they do not focus on the reduction (or the modeling) of the delay penalty and link stress, nor on large groups. Other application level multicast architectures include ScatterCast [4], Yoid [11], ALMI [22], and Overcast [15]. These architectures either optimize delay or optimize bandwidth.

More recently, Content-Addressable Network (CAN)-based multicast [24] was proposed to partition member nodes into bins using proximity information obtained from DNS and delay measurements. Node degree constraints and diameter bounds in the constructed overlay multicast network are investigated in [27]. Liebeherr et al. investigate Delaunay triangulations for routing in overlay networks in [18]. A prefix-based application level routing protocol is used in Bayeux [30]. Hierarchical approaches to improve scalability are also currently being investigated by several researchers. A protocol that was theoretically proven to build low diameter and low degree peer-to-peer networks was recently described in [20].

In addition to overlay multicast proposals, several recent studies are related to the TAG approach. A unicast-based protocol for multicast with limited router support (that includes some ideas that inspired TAG) is the REUNITE protocol [28]. Overlay networks that detect performance degradation of current routing paths and re-route through other hosts include Detour and RON [2]. Jagannathan and Almeroth [14] propose an algorithm which uses multicast tree topology information (similar to the manner in which we exploit path information in TAG) and loss reports from receivers for multicast congestion control.

6. CONCLUSIONS AND FUTURE WORK

We have designed and studied a topology aware application level multicast protocol, TAG. Network topology information is used to construct an overlay network with low delay penalty and a limited number of duplicate packets. Bandwidth is also considered in tree construction as a secondary metric. We have studied the properties of TAG, and analyzed its economies of scale factor, compared to both unicast and IP multicast. Simulation results on the Transit-Stub model (GT-ITM), Inet, and NLANR data indicate the effectiveness of the algorithm in building efficient trees for a large number of group members.

We are currently generalizing our economies of scale analysis to more general trees, and to other overlay tree construction algorithms. Also, we are studying the adaptivity of the TAG algorithm to network and group changes. We are also investigating the use of partial topology and bandwidth data in a partial path subsequence matching algorithm, with dynamically varying values of k. In addition, we will incorporate TAG into two different applications (e.g., a multi-player online game and a video streaming application), and conduct experiments for evaluating the practical aspects of TAG implementation, tuning, and performance in the Internet.

7. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments. This research is sponsored in part by the Purdue Research Foundation, and the Schlumberger Foundation technical merit award.

8. REFERENCES

[1] C. Adjih, L. Georgiadis, P. Jacquet, and W. Szpankowski. Multicast Tree Structure and the Power Law. In *Proc. of SODA*, 2002.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. of ACM SOSP*, October 2001.

[3] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT): An architecture for scalable multicast routing. In *Proceedings of the ACM SIGCOMM*, August 1993.

[4] Y. Chawathe, S. McCanne, and E. A. Brewer. An Architecture for Internet Content Distribution as an Infrastructure Service. Ph.D. Thesis, Fall 2000.

[5] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proc. of ACM SIGCOMM*, August 2001.

[6] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *Proc. of ACM Sigmetrics*, June 2000.

[7] J. Chuang and M. Sirbu. Pricing Multicast Communications: A Cost-Based Approach. In *Proc. of Internet Society INET*, July 1998.

[8] S. Deering and D. Cheriton. Multicast routing in datagram inter-networks and extended lans. *ACM Trans. on Computer Systems*, 2(8):85–110, May 1990.

[9] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network Magazine*, January/February 2000.

[10] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proc. of ACM SIGCOMM*, pages 251–262, August 1999.

[11] P. Francis. Yoid: Your Own Internet Distribution, April 2000. http://www.aciri.org/yoid/.

[12] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: a global internet host distance estimation service. *IEEE/ACM Transactions on Networking*, October 2001.

[13] V. Jacobson. Pathchar. http://www.caida.org/tools/utilities/others/pathchar/.

[14] S. Jagannathan and K. Almeroth. Using Tree Topology for Multicast Congestion Control. In *Proc. of International Conference on Parallel Processing*, September 2001.

[15] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O. Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. of OSDI*, October 2000.

[16] C. Jin, Q. Chen, and S. Jamin. Inet: Internet Topology Generator. Technical Report CSE-TR-443-00, Univ. of Michigan, 2000.

[17] K. Lai and M. Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *Proc. of USENIX Symposium on Internet Technologies and Systems*, March 2001.

[18] J. Liebeherr and M. Nahas. Application-layer Multicast with Delaunay Triangulations. In *Proc. of IEEE GLOBECOM*, November 2001.

[19] NLANR. National Laboratory for Applied Network Research, 2000. http://moat.nlanr.net/Routing/rawdata.

[20] G. Pandurangan, P. Raghavan, and E. Upfal. Building Low-Diameter P2P Networks. In *Proc. of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*, 2001.

[21] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. of ACM SIGCOMM*, pages 25–38, August 1996.

[22] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: an Application Level Multicast Infrastructure. In *Proc. of USENIX Symposium on Internet Technologies*, March 2001.

[23] G. Phillips, S. Shenker, and H. Tangmunarunkit. Scaling of multicast trees: Comments on the Chuang-Sirbu scaling law. In *Proc. of ACM SIGCOMM*, 1999.

[24] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proc. of IEEE INFOCOM*, June 2002.

[25] D. Senie. Network Address Translator (NAT)-Friendly Application Design Guidelines. RFC 3235, January 2002.

[26] A. Shaikh, M. Goyal, A. Greenberg, R. Rajan, and K. K. Ramakrishnan. An OSPF Topology Server: Design and Evaluation, 2001. http://www.cis.ohio-state.edu/∼mukul/research.html.

[27] S. Shi and J. Turner. Routing in Overlay Multicast Networks. In *Proc. of IEEE INFOCOM*, June 2002.

[28] I. Stoica, T. S. E. Ng, and H. Zhang. REUNITE: A Recursive Unicast Approach to Multicast. In *Proc. of IEEE INFOCOM*, 2000.

[29] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM*, 1996.

[30] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *Proc. of ACM NOSSDAV*, June 2001.