Topology-Aware Overlay Networks for Group Communication

Minseok Kwon
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–2066
kwonm@cs.purdue.edu

Sonia Fahmy
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–2066
fahmy@cs.purdue.edu

ABSTRACT

We investigate a heuristic application-level (overlay) multicast approach, which we call Topology Aware Grouping (TAG). TAG exploits underlying network topology data to construct multicast overlay networks. TAG uses information about *overlap in routes to the sender* among group members to set up the overlay network in a distributed low-overhead manner. The constructed tree has low relative delay penalty, and introduces a limited number of identical copies of a packet on the same link– assuming underlying routes are of good quality. We study the properties of TAG, and quantify its economies of scale factor, compared to unicast and IP multicast. We also compare TAG with End System Multicast (ESM) in a variety of simulation configurations, including both real Internet topologies and generated topologies. Our results indicate the effectiveness of our heuristic in reducing delays and duplicate packets, with reasonable time and space complexities. TAG can be combined with delay and bandwidth bounds to construct overlays that satisfy application requirements.

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 [14, 15]. Application-level multicast approaches using overlay networks [9, 11, 12, 18, 24, 32, 42] have been recently proposed as a viable alternative to IP multicast. In particular, End System Multicast (ESM) [11, 12] has gained considerable attention due to its success in conferencing applications. The main idea of ESM (and its Narada protocol) is that end systems exclusively handle group management, routing information exchange, and overlay forwarding tree construction. The efficiency of large-scale overlay multicast trees, in terms of both performance and scalability, is the primary subject of this paper.

We investigate a simple heuristic, which we call Topology Aware Grouping (TAG), to exploit underlying network topology data in constructing efficient overlays for application-level multicast. Our heuristic works well when underlying routes are of good quality, e.g., in intra-domain environments, and when final hop delays are small. Each new member of a multicast session first determines the path (route) from the root (primary sender) of the session to itself. The overlap among this path and other paths from the root is used to partially traverse the overlay data delivery tree, and determine the best parent and children for the new member. The constructed overlay network has a low delay penalty and limited duplicate packets sent on the same link. TAG nodes maintain a small amount of state information– IP addresses and paths of only their parent and children nodes.

Unlike ESM, the TAG heuristic caters to applications with a large number of members, which join the session at different times. TAG works best with applications which 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. Bandwidth is also used 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 also demonstrate via simulations the effectiveness of TAG in terms of delay, number of identical 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, its extensions, and some design considerations. 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 overlay multicast has emerged as a practical alterna-

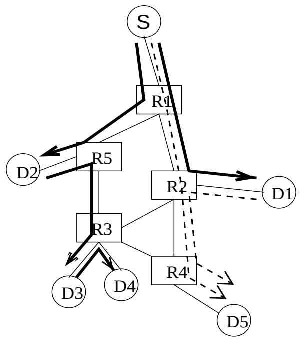


Figure 1: Example of topology aware overlay networks

tive to IP multicast, overlay network performance in terms of delay penalty and number of identical 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 multicast 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, assuming the underlying routes are of good quality. By "underlying network topology," we mean the shortest path information that 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, or according to administrative policies. Using topology information is illustrated in figure 1. In the figure, source S (the root node) and destinations $D1$ to $D4$ are end systems that belong to the multicast group, and $R1$ to $R5$ are routers. Thick solid lines denote the current data delivery tree from S to $D1-D4$. The dashed lines denote the shortest paths to a new node $D5$ from S and $D1$. If $D5$ wishes to join the delivery tree, which member is the best parent node to $D5$? If $D1$ becomes the parent of $D5$, a relay path from $D1$ to $D5$ is consistent with the shortest path from S to $D5$. Moreover, no duplicate packet copies are necessary in the sub-path S to $R2$ (packets in one direction are counted separately from packets in the reverse direction). This heuristic is similar to determining if a friend is, more or less, on your way to work, so giving him/her a ride will not excessively delay you, and you can reduce overall traffic by car pooling. If he/she is out of your way, however, you decide to drive separately. In addition, this heuristic is subject to both capacity (e.g., space in your car) and latency (car pooling will not make the journey excessively long) constraints.

Of course, it is difficult to determine the shortest path and the number of identical packets, in the absence of any knowledge of the underlying network topology. If network topology information can be obtained by the multicast participant (as discussed in section 2.6), nodes need not exchange complete end-to-end measurements, and topology information can be exploited to construct high quality overlay trees. Therefore, our heuristic is: a TAG destination selects as a parent the destination whose shortest path from the source has maximal overlap with its own path from the source. This heuristic minimizes the increase in number of hops (and hence delay if we assume low delay of the last hop(s)) over the shortest unicast path. We also use loose bandwidth constraints.

As with all overlay multicast 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 primary sender in a session. The primary sender serves as the root of the multicast delivery tree. The case of multiple senders will be discussed in section 2.8.

2.1 Assumptions

TAG makes a number of assumptions:

1. **TAG is used for single-source multicast or core-based multicast:** The source node or a selected core node is the root of the multicast forwarding tree (similar to single-source multicast [21] or core-based multicast [6] for IP multicast).

2. **Route discovery methods exist:** TAG can obtain the shortest path for a sender-receiver pair on the underlying network. A number of route discovery tools are discussed in section 2.6.

3. **All end systems are reachable:** Any pair of end systems on the overlay network can communicate using the underlying network. Recent studies, however, indicate that some Internet routes are unavailable for certain durations of time [31, 26, 7].

4. **Underlying routes are of good quality (in terms of delay):** Intra-domain routing protocols typically compute the shortest path in terms of delay for a sender-receiver pair. Recent studies, however, indicate that the current Internet demonstrates a significant percentage of routing pathologies [31, 36]. Many of these arise because of policy routing techniques employed for inter-domain routing. TAG is best suited for well-optimized routing domains.

5. **The last hop(s) to end systems exhibit low delay:** A long latency last hop to an end system, e.g., a satellite link, adversely affects TAG performance. TAG works best with low-delay a last hop to an end system (or last few hops for the partial path matching flavor of TAG, as discussed in section 2.5).

2.2 Definitions

We define the following terms, which will be used throughout the paper.

DEFINITION 1. *A $path$ from node A to node B in TAG, 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.*

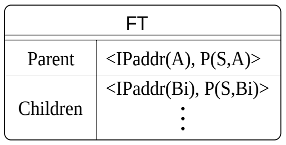
DEFINITION 2. *$A \succ B$ if $P(S, A)$ is a prefix of $P(S, B)$, where S is the root of the tree.*

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

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.

2.3 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 best parent for a new node N in a tree rooted at S is:



FT	
Parent	<IPaddr(A), P(S,A)>
Children	<IPaddr(Bi), P(S,Bi)> ⋮

Figure 2: Family table (FT)

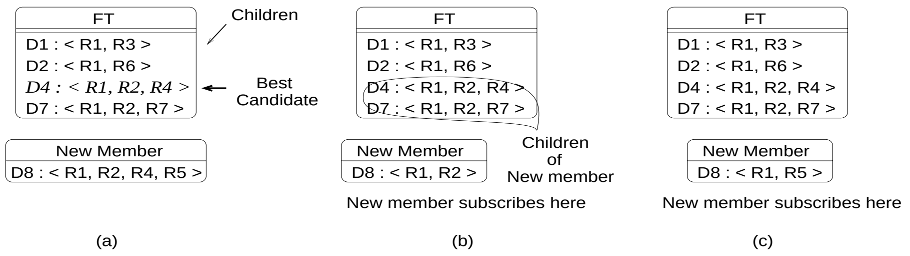


Figure 3: The three conditions for complete path matching

A node C ($C \neq N$) in the tree, such that $C \succ N$ and $len(P(S,C)) \geq len(P(S,R))$ for all nodes $R \succ N$ in the tree.

The algorithm considers three mutually exclusive conditions, as depicted in figure 3. Let N be a new member wishing to join a current session. Let C be the node being examined, and S be the root of the tree. 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 its children (figure 3(b)). In case no child of C satisfying the first or second conditions exists, N becomes a child of C (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 is the node currently being examined by N; and $target$ is the next node which N will probe, if necessary.

There are two reasons for selecting a node (in the first and second conditions) whose *spath* is the longest prefix of the *spath* of 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 the additional delay introduced by overlays. Second, sharing the longest prefix curtails the number of identical packet copies in the overlay network, since a single packet is generated over the shared part.

2.4 Tree Management

In this section, we discuss the multicast tree management protocol, including member join and member leave operations, and fault resilience issues.

2.4.1 Member Join

A new member joining a session sends a JOIN message to the primary 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. The FIND 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 (showing only the children) is given next to it. The destinations join the session in the order $D1$ to $D5$. Upon the receipt of a JOIN message from

```
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

$D1$, 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 $D1$ to $D4$ are updated accordingly (figure 6(d)). Finally, $D5$ joins the session as a child of $D4$ (figure 6(e)). Figure 6(e) depicts the final state of the multicast forwarding tree and the FT at each node.

2.4.2 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 (fig-

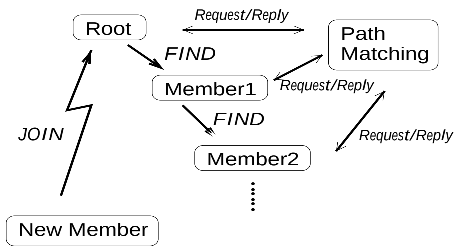


Figure 5: A member join process in TAG

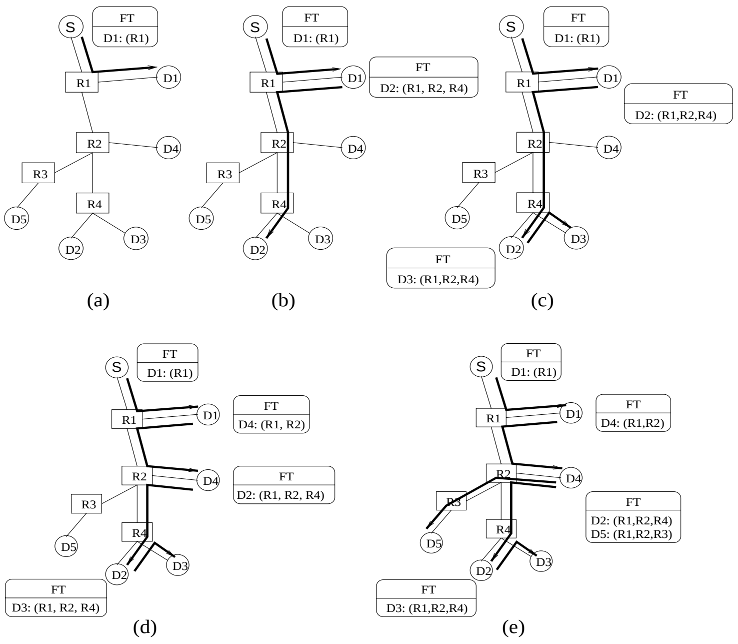


Figure 6: Member join in TAG

ure 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 updated multicast forwarding tree is illustrated in figure 7(b).

2.4.3 Fault Resilience

Failures in end systems participating in 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.

2.5 Bandwidth Considerations

Since TAG targets delay-sensitive applications, delay (as defined by the underlying IP routing protocol) is the primary metric used in path matching. The complete path matching algorithm presented in figure 4 can reduce the delay from source to destination, and reduce the total number of identical packets. However, high link stress [12] (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

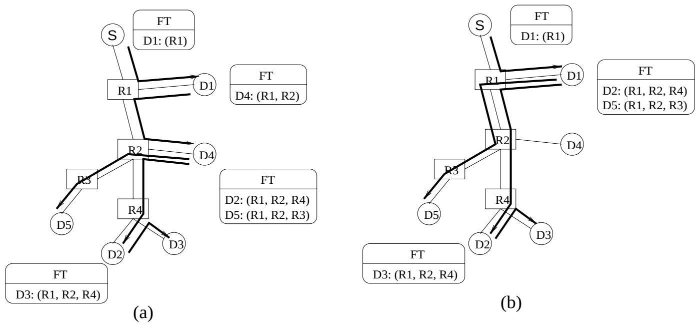


Figure 7: Member leave in TAG

attach to a node A as a child, if A has a common *spath* prefix of length $len(P(S, A)) - k$ with B (S is the root of the tree), even if the remaining k elements of the *spath* of A do not match the *spath* of B. We call this method partial path matching or minus-k path matching. We use the symbol $\succ_{partial(k)}$ to denote minus-k path matching.

Minus-k 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 give 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 to a node, but it reduces the maximum stress a node may experience, therefore increasing available bandwidth to the node.

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 which are 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 gives the maximum bandwidth among these nodes is the one selected. The partial path matching algorithm is presented in figure 8. The member leave operation also employs minus-k path matching. When the parent of a leaving member receives a LEAVE message, the parent first removes the leaving member from its FT. Children of the leaving member (included in the LEAVE message) then execute minus-k path matching at the parent of the leaving member to find their new parents. Figure 9 illustrates an example of member join and leave with $k = 1$. A new member $D3$ takes $D1$ as its parent since $D1$ provides more bandwidth than $D2$. When $D1$ leaves, $D4$ becomes a child of $D3$. $D3$ maximizes bandwidth to $D4$ among the children of $D0$ ($D2$ and $D3$).

Table 1 shows the tradeoff between the delay (mean RDP as defined in section 4), total link stress on the tree, and maximum link stress in the tree, for a variety of $bwthresh$ values (in kbps). 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 identical 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 identical packets and high maximum stress, yielding little bandwidth for many connections.

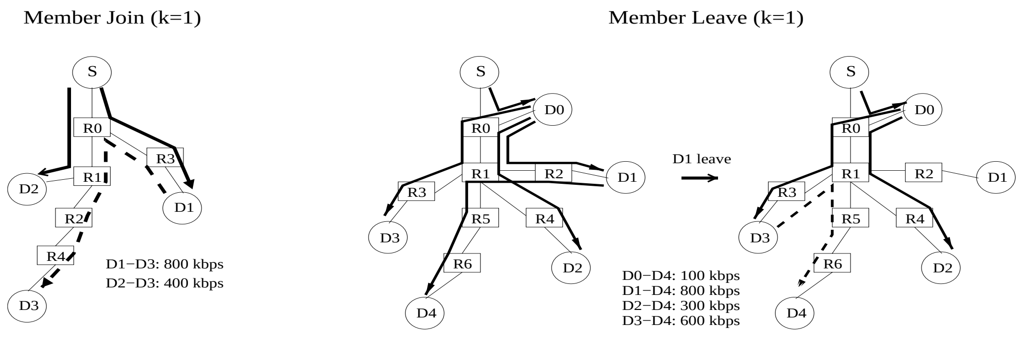


Figure 9: Member join and leave with partial path matching

Table 1: Tradeoffs with different $bwthresh$ values (in kbps)

$bwthresh$ (in kbps)	RDP	Total stress	Max. stress
400	1.570779	1804	106
300	1.559278	1928	104
200	1.526061	2142	105
100	1.384193	2680	146
50	1.319069	2578	189
20	1.301828	3732	413
0	1.352894	3992	340

2.6 Obtaining 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 for performing this. The first approach is to use a network path-finding tool such as *traceroute*. *Traceroute* has been extensively used for network topology discovery [19, 31]. Some routers, however, do not send ICMP Time-Exceeded messages when Time-To-Live (TTL) reaches zero for several reasons, the most important of which is security. Recently, we conducted simple experiments where we used *traceroute* for 50 sites in different continents at different times and on different days. Approximately 82−90% of the routers responded. The average time taken to obtain and print the entire path information was 5.2 seconds, with a maximum of 42.6 seconds and a minimum of 0.2 seconds. 5−8% *traceroute* failures were reported in [31]. A recent study [20] indicates that router ICMP generation delays are generally in the sub-millisecond range ($<$ 500 $\mu secs$). This shows that only a few routers in today's Internet are slow in generating ICMP Time-Exceeded messages.

The second option is to exploit topology servers. For example, an OSPF topology server [38] can track intra-domain topology, either by listening to OSPF link state advertisements, or by pushing and pulling information from routers via SNMP. Network topology can also be obtained from periodic dumps of router configuration files [17], from MPLS traffic engineering frameworks [4], and from policy-based OSPF monitoring frameworks [5]. Internet topology discovery projects [16, 8, 10] can also supply 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 [29, 16, 8, 10]. Partial information can still be exploited by TAG for partial path matching of longest common subsequences.

Bandwidth estimation tools are important for TAG, in conjunction with in-band measurements, to estimate the available bandwidth between nodes under dynamic network conditions. Tools similar to *pathchar* [22] estimate available bandwidth, delay, average queue, and loss rate of every hop between any source and destination on the Internet. *Nettimer* [27] is useful for low-overhead measurement of per-link available bandwidth. Other bandwidth or throughput measurement tools are linked through [1].

2.7 Adaptivity and Scalability

If network conditions change and the overlay tree becomes inefficient (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 of that group are updated. Third, when changes in part of the *spath* are detected for a destination, not only the destination being probed, but also all the destinations in the same group and all the destinations in groups with overlapping *spaths*, are updated.

Although these TAG reorganizing mechanisms help reduce overhead, the root is likely to experience a high load when a large number of members join or rejoin simultaneously. The root is also a single failure point. To address these limitations, mechanisms similar to those used in Overcast [24] can be used. For example, requests from members to the root are redirected to less-burdened replicated roots.

Another important consideration is the quality of the TAG tree in terms of both delay and bandwidth. TAG aligns overlay routes with underlying routes, assuming underlying routes are of good quality (fourth assumption in section 2.1). Unfortunately, today's Internet routing, particularly inter-domain routing, exhibits pathologies, such as slow convergence and node failures. Savage *et al.* [36, 35] found that there is an alternate path with significantly superior quality to the IP path in 30−80% of the cases (30−55% for latency, 75−85% for loss rate, and 70−80% for bandwidth). Intra-domain network-level routes, however, are generally of good quality, and, in the future, research on inter-domain routing and traffic

```
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 $\succ_{partial(k)}$ N && bandwidth(ch) > maxbw)
          then
              target := ch;
              maxbw := bandwidth(ch);
              flag := condition(1); fi;
        if (N $\succ$ ch && flag is NOT condition(1))
          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

engineering may improve route quality in general. Another important consideration is that a long latency last hop(s) to a TAG parent node may yield high delay to its children (section 2.1). A delay-constrained overlay tree construction mechanism can be combined with the TAG heuristic to prevent such high delay paths.

As for bandwidth, it is only considered as a secondary metric (as a tie breaker among equal delay paths) in tree construction (section 2.5). Minus-k path matching does not guarantee bandwidth—it simply explores more potential paths when bandwidth is scarce. A bandwidth-constrained tree construction mechanism can be incorporated into TAG if bandwidth guarantees are required.

2.8 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 session. This approach is suitable for mostly single-sender applications, where the primary 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. This is similar to the core selection problem in core-based tree approaches for multicast routing [6]. Multiple (backup) roots are also important for fault tolerance.

3. ANALYSIS OF TAG

In this section, we investigate the properties of TAG, and study its bandwidth penalty compared to IP multicast. For simplicity, we use TAG with complete path matching (figure 4) in our analysis, except for the complexity analysis where we analyze both complete and mins-k path matching.

3.1 Properties of TAG

In this section, we study the conditions used in the path matching algorithm, and the 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 by 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 parent) of node B. In this case, there must be n ($n > 0$) nodes such that $M_1 = Parent(B)$, $M_2 = Parent(M_1)$, . . ., $M_n = Parent(M_{n-1})$, $A = Parent(M_n)$. $M_1 = Parent(B) \succ B$ 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. □

In figure 1, 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: We first prove mutual exclusion. To show mutual exclusion 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 both hold at the same time. 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 previously ensured that B' is a descendant 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.

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

Proof: By lemma 1, a unique relationship among every two nodes (i.e., parent, ancestor, child, descendant, or none) is established

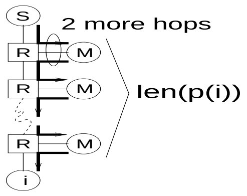


Figure 10: Bound on the number of hops in TAG

among every two nodes which have different *spaths*, 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 now study the properties of a parent node.

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

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 $len(P(S,A)) > len(P(S,B))$, then $P(S,B)$ is a prefix of $P(S,A)$. By definition 2, $B \succ A$. Therefore, A must be a descendant of B according to lemma 1. The path matching algorithm, however, would make C 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 bound for the number of hops on the path from the root to each member.

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

Proof: Consider $P(S,i)$, the path from root S to i. By the definition of len, $SPD(i) = len(P(S,i)) + 1$ since $SPD(i)$ is the number of hops on the path $P(S,i)$. The fact that $SPD(i)$ is the number of hops on the shortest path $P(S,i)$ ensures $SPD(i) \le E(i)$. The maximum $E(i)$ occurs when i has as many ancestors as $len(P(S,i))$. This situation is depicted in figure 10. In the figure, R denotes a router; i is a destination, and nodes M are all ancestors of i. For every M, 2 hops are added to the path. Thus, $2 \times len(P(S,i))$ hops are added to $SPD(i)$. Therefore, the maximum $E(i)$ is $3 \times SPD(i) - 2$. □

3.2 Time Complexity

In this section, we analyze the time complexities of both complete path matching and minus-k path matching. For simplicity, we assume that an overlay multicast tree has n end systems, and that each end system has an average of m children. We also assume that an average of v routers exist over the link between a parent and its child. After the source discovers the path to a new member, the member join process requires: (1) operations for each node, and (2) tree traversal. For (1), suppose that the new member has matched its path to a node α at level $i-1$ and is searching for a matching node at level i in the overlay multicast tree, where $0 < i < \log_m n$. (The height of the tree is $\log_m n$, which is assumed to be an integer.) In complete path matching, operations in (1) require mvi operations for the new member at α to search for the next matching node at level i, in the worst case. The new member can find the next matching node at level i by examining at most vi routers per child for each of the m children of α. Operation (2) is of order $\log_m n$ from the root to a leaf node. Therefore, the time complexity of member join is:

$$\sum_{i=1}^{\log_m n} mvi = \frac{mv \log_m n(\log_m n + 1)}{2} = O(mv \log_m^2 n) \quad (1)$$

Member leave requires one deletion and m additions of FT entries. Each entry requires $v \log_m n$ operations for the worst case path length. Thus, the time complexity of member leave is $O(mv \log_m n)$.

In minus-k path matching, (1) requires $m(k + vi)$ operations if $k + vi \le v \log_m n$. Otherwise, (1) requires $mv \log_m n$ operations. A new member matched with node α examines k more routers than in the complete path matching case. Hence, $k + vi$ routers are examined by the new member per child to find the next matching node if $k + vi \le v \log_m n$. Otherwise, $v \log_m n$ routers are examined (the maximum path length). This is performed for each of the m children of α. Since operation (2) requires $\log_m n$ operations and assuming that k is small, the time complexity of member join is:

$$\begin{aligned} & \sum_{i=1}^{\lfloor \log_m n - \frac{k}{v} \rfloor} m(k+vi) + \sum_{i=\lfloor \log_m n - \frac{k}{v} \rfloor + 1}^{\log_m n} mv \log_m n \\ = \;& m(\lfloor \log_m n - \frac{k}{v} \rfloor) \left(\frac{v(\lfloor \log_m n - \frac{k}{v} \rfloor + 1)}{2} + k \right) \\ + \;& km \log_m n \\ = \;& O(mv \log_m^2 n) \qquad (2) \end{aligned}$$

As discussed in section 2.5, children of a leaving member rejoin the session starting from the parent of the leaving member in minus-k path matching. In this process, each of m children of the leaving member requires $O(mv \log_m^2 n)$ operations for the rejoin. Hence, the time complexity of member leave is $O(m^2 v \log_m^2 n)$ in this case.

3.3 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 naive unicast; and (2) how much additional bandwidth it consumes compared to IP multicast. IP multicast savings over naive unicast have been studied in [2, 13, 33]. Chuang and Sirbu [13] investigated the cost of IP multicast for a variety of real and generated network topologies. Their work was motivated by multicast pricing. They found that $L(m) \propto m^{0.8}$, where $L(m)$ is the ratio between total number of multicast links and average unicast path length, and m is the number of distinct routers to which end systems are connected. They also found that the cost of a multicast tree saturates when the number of subscribing end systems exceeds a certain value. Based on these results, they suggested membership-based pricing until m reaches the saturation point, and flat-rate pricing beyond that point.

In this section, we quantify the network resources consumed by TAG. We derive a bound for the function $L_{TAG}(n)$, which denotes the sum of link stress values on all router-to-router links, for a multicast group of size n. Although the number of distinct routers to which end systems are connected is used in [13], we use

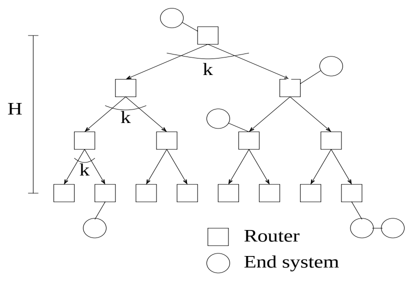


Figure 11: A k-ary Tree Model

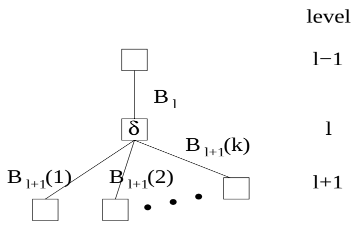


Figure 12: TAG Trees

n, the number of end systems in a multicast group. As discussed in [2], using the number of end systems is intuitively appealing and makes the analysis simpler. Note that m can be approximated by $m = M(1-(1-\frac{1}{M})^n)$ where M is the total number of possible routers to which end systems can be connected. $m \approx n$ when $\frac{n}{M} \ll 1$.

For simplicity, we assume a k-ary data dissemination tree in which tree nodes denote routers (as in [2, 33]), as depicted in figure 11. The height of the tree is H and all nodes except the leaves have degree k. We assume that no unary nodes (nodes at which no branching occurs) exist. Therefore, our results are approximate. An end system can be connected to any router (node in the tree). Suppose that n end systems join the multicast session. The probability that at least one end system is connected to a given node is:

$$p_n = 1 - (1 - \frac{1}{M})^n \tag{3}$$

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

We now evaluate the cost of transmission at each level of the tree. In figure 12, B_l indicates the cost over the link between node δ at level l and its parent at level $l-1$, and $B_{l+1}(a)$ denotes the cost over the links between node δ at level l and its children at level $l+1$ for $1 \le a \le k$. We compute B_l considering two different cases: when at least one end system is connected to node δ, and when no end system is connected to δ. Let B_l^1 be the cost in the first case and B_l^2 be the cost in the second case. TAG enforces that the first case costs one, for transmission between node δ and its parent. This is because node δ sends packets from the parent to the children ($B_l = 1$). In the second case, however, since no end system relays the packets at δ, the cost over outgoing links of δ towards the leaves is equal to the cost over the link between δ and its parent. Therefore, $B_l^1 = p_n$ and $B_l^2 = (1-p_n)\sum_{a=1}^{k} B_{l+1}(a)$. We assume the end systems are uniformly distributed to tree 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, $E[B_l^2] = (1-p_n)kE[B_{l+1}]$.

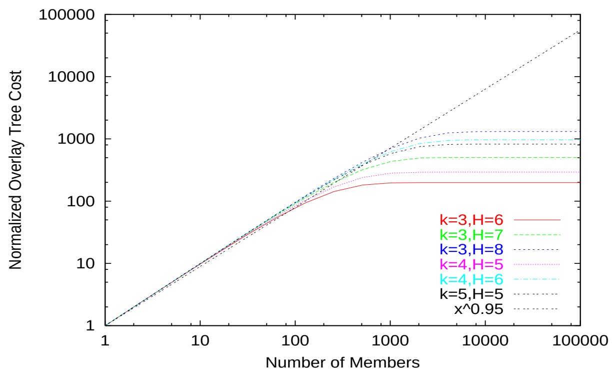


Figure 13: TAG tree cost versus group size

Hence, $E[B_l]$ is defined as follows:

$$E[B_l] = E[B_l^1] + E[B_l^2] = p_n + (1-p_n)kE[B_{l+1}] \tag{4}$$

$$E[B_H] = p_n \tag{5}$$

Solving the recurrence in (4) and (5), we obtain:

$$\sum_{l=1}^{H} k^l E[B_l] = \frac{k^{H+1}(1-p_n)\{1-(1-p_n)^H\}}{k(1-p_n)-1} - \frac{p_n(k^{H+1}-k)}{\{k(1-p_n)-1\}(k-1)} \tag{6}$$

The cost $L_{TAG}(n)$ is given by (6):

$$L_{TAG}(n) = \sum_{l=1}^{H} k^l E[B_l] \tag{7}$$

Figure 13 plots the normalized overlay tree cost of TAG for a variety of k and H values on a log-log scale. The normalized overlay tree cost $L_{TAG}(n)/\hat{u}$ is defined as $L_{TAG}(n)$, the cost of an overlay with n members, divided by the average number of hops (only counting router-to-router links) for a unicast path from the source to receivers, $\hat{u}$. Since we assume end systems are uniformly distributed at nodes, $\hat{u}$ is the average number of hops from the root to a node on the overlay tree:

$$\hat{u} = \frac{1}{M}\sum_{i=1}^{H} ik^i \tag{8}$$

All curves stabilize for group sizes exceeding $1000-5000$ members. The slope decreases because as group size grows, more end systems can share the links yielding more bandwidth savings. This is an important advantage of TAG over unicast. The figure shows that, approximately, $L_{TAG}(n)/\hat{u} \propto n^{0.95}$ before the curves stabilize. The factor 0.95 is smaller than unicast, but larger than the factor for IP multicast ($L_{IPmulticast}(n)/\hat{u} \propto n^{0.8}$), 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 [12] to evaluate and compare their performance. The simulators model propagation delays, but not queuing delays and packet losses. Two sets of simulations were performed on different topologies. The first set uses Transit-Stub topologies generated by GT-ITM [41]. The Transit-Stub model generates router-level Internet topologies. We also simulate AS-level topologies, specifically the actual Internet AS topologies from NLANR [29] and topologies generated by Inet [25].

The Transit-Stub model generates 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. When the total is 492 nodes, there are 2 transit domains, 6 nodes per transit domain, 5 stub domains per transit node, and 8 nodes per stub domain. Similar distributions are used when the total number of nodes is 984 and 1640. We label the 3 transit-stub topologies 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 members. GT-ITM generates symmetric link delays ranging from 1 to 55 ms for transit-transit or transit-stub links. We use 1 to 10 ms delays within a stub. We randomly assign bandwidth ranging between 100 Mbps and 1 Gbps to backbone links. We use 500 kbps to 10 Mbps for the links from edge routers to end systems.

The AS topologies from NLANR and Inet give AS-level connectivity. AS-level maps have been shown to exhibit a power-law [16]. This means that a few nodes have high-degree connectivities, while most other nodes exhibit low-degree connectivities. We use the 1997 and 1998 NLANR data sets, named AS97 and AS98, respectively. We also use the 1997 Inet data set (named Inet97) and the 1998 Inet data set (Inet98), which have the same number of ASes as the NLANR data sets: 3015 and 3878. We have 4000 (for 1997) and 5000 (for 1998) members in a 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 configurations. The link delays are asymmetric.

We assume that the IP layer routing algorithm uses delay as a metric for finding shortest paths. The routing algorithm for the mesh in ESM uses discretized levels of available bandwidth (in 200 kbps increments) 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 kbps, unless otherwise specified. We use the same parameters for ESM used in the simulations in [11, 12] (lower degree bound = 3, upper degree bound = 6, high delay penalty = 3), except for delay-related parameters (close neighbor delay = 85 ms) since we assign a wider range of delays to the links.

We use the following performance metrics [11, 12] for evaluating TAG and ESM:

1. **Mean Relative Delay Penalty (RDP):** RDP is the relative increase in delay between the source and a receiver in TAG against unicast delay between the source and the same receiver. The RDP from source s to receiver d_r is the ratio $\frac{latency(s,d_r)}{delay(s,d_r)}$. The latency $latency(s, d_r)$ from s to d_r is defined to be $delay(s, d_0) + \sum_{i=0}^{l-1} delay(d_i, d_{i+1}) + delay(d_l, d_r)$, assuming s delivers data to d_r via the sequence of end systems $(d_0, \cdots, d_l)$. Here, $delay(d_i, d_{i+1})$ denotes the end-to-end delay from d_i to d_{i+1}. We compute the mean RDP of all receivers.

2. **Link Stress:** Link stress is the total number of identical copies of a packet over a physical link. We compute the total stress for all tree links. We also compute the maximum value of link stress among all links.This is clearly a network-level metric and is not of importance to the application user.

3. **Mean Available Bandwidth (in kbps):** This is the mean of the available bottleneck bandwidth between the source and all receivers.

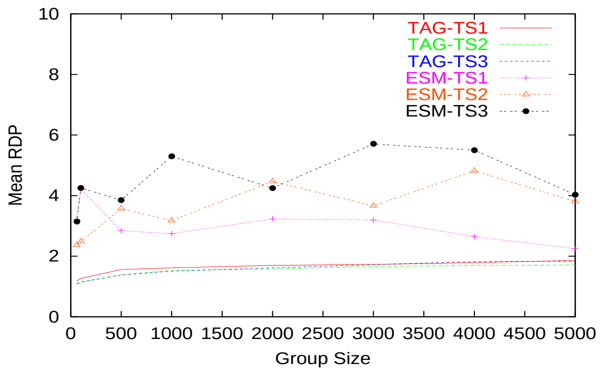


Figure 14: Mean RDP: TAG versus ESM

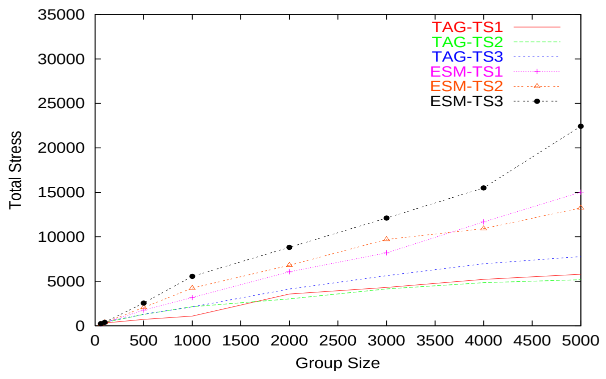


Figure 15: Total stress: TAG versus ESM

The TAG tree cost is also computed in section 4.3, and compared to IP multicast and unicast cost.

4.2 Performance Results

4.2.1 Transit-Stub Topologies

The mean RDP values for TAG and ESM on the three different Transit-Stub topologies (TS1, TS2, TS3) are plotted in figure 14. 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, and TS3 are all in the range of 1 to 2, while the mean RDP values for ESM range from 2 to 6. This is because TAG considers delay as the primary metric while ESM uses delay only as a tie breaker. 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. As more end systems join in TAG, the mean RDP values increase due to the bandwidth constraint in partial path matching (even though lower latency paths may become available). We observe that, unlike TAG, the mean RDP values for ESM do not always increase with the increase in group size.

Figure 15 illustrates the total stress of TAG and ESM for the three different topologies. For all group sizes and topologies, TAG

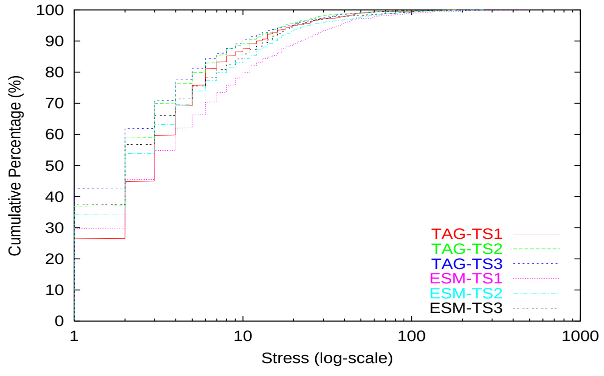


Figure 16: Cumulative distribution of stress

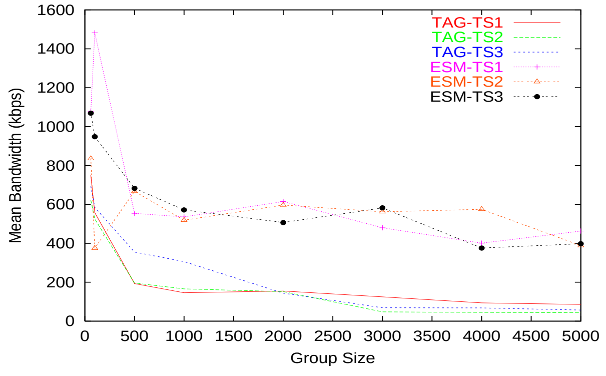


Figure 18: Mean bandwidth: TAG versus ESM

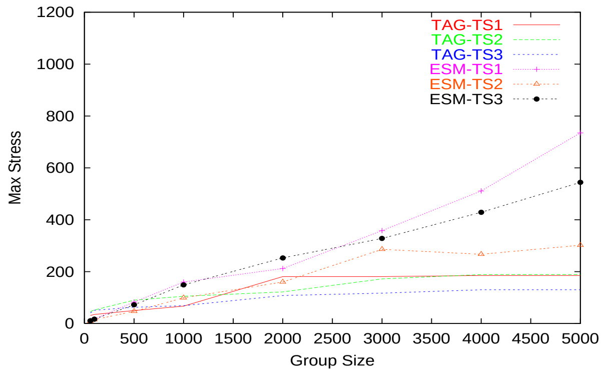


Figure 17: Maximum stress: TAG versus ESM

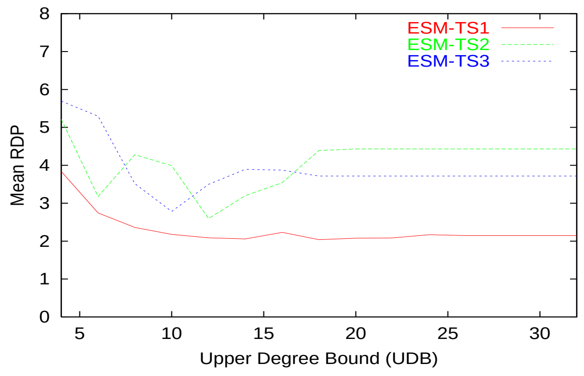


Figure 19: Mean RDP for different UDB values in ESM

total stress is below 8000. In contrast, ESM exhibits higher stress. The total stress for all 6 configurations increases in proportion to the group size, since more identical packets traverse physical links when more end systems join the session. TAG-TS1 and TAG-TS2 exhibit the lowest total stress. TAG can avoid some duplicate packets since TAG aligns overlay and underlying routes, subject to bandwidth availability. Packets from the source to a receiver are not duplicated along the path from the source to the parent of that receiver. Figure 16 depicts the cumulative distribution of total stress. The figure shows that the three TAG configurations have slightly more low-stress links than the three ESM configurations.

Figure 17 illustrates that TAG, with the correct parameters, can reduce the maximum stress value as well. With the complete path matching algorithm, a strategically located end system attached to a high-degree router can be the parent of numerous nodes, which severely constrains the bandwidth available to each of these nodes, and increases the stress at this end system. The minus-k path matching algorithm remedies this weakness, as shown in the figure.

The mean bandwidth, depicted in figure 18, denotes the average of the available bottleneck bandwidths from the source to all members. The available bottleneck bandwidth with ESM is high (from 600 to 1600 kbps) for up to 500 members, and then stabilizes at approximately 600 kbps for groups exceeding 500 members. TAG gives $200-800$ kbps bandwidth for up to 500 members. For larger groups, the bandwidth rapidly drops to under 200 kbps. The bottleneck bandwidth given by TAG continues to decrease as the group size increases. TAG bottleneck bandwidth is very sensitive to the number of members, and to the bandwidth threshold ($bwthresh$ = 200 kbps here). This and the fact that ESM optimizes bandwidth explain why ESM performs better than TAG.

An important point to note is that in the ESM algorithm, the lower and upper degree bounds (LDB and UDB, respectively) for each group member play a key role. The two parameters control the number of neighbors that each member communicates with. In particular, the upper degree bound is significant, as it impacts both protocol scalability and performance. In our simulations, we observe that increasing the upper degree bound for ESM reduces the delay penalty in some cases, but not always. Figure 19 plots the mean RDP values of ESM versus different UDB values on the 3 topologies TS1, TS2, and TS3 for 1000 members. In the figure, the mean RDP values decrease as UDB increases for ESM-TS1 and ESM-TS3 (except for an increase between UDB=10 and UDB=15). The mean RDP stabilizes beyond a certain UDB value. Increasing UDB generally helps a member find the best paths in terms of delay penalty and bandwidth. However, due to the discretized levels of available bandwidth used as a primary metric in ESM, changes using different UDB values are not substantial. A higher UDB, of course, increases the volume of routing information exchanged, which is detrimental to scalability. We have observed no significant change in the mean bandwidth, total stress, or maximum stress, with higher UDB values.

The parameter choices for TAG, most significantly $bwthresh$ (which should be tuned according to application bandwidth requirements), 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 2.5.

4.2.2 AS and Inet Topologies

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

Configuration	Algorithm	Mean RDP	Total link stress	Max. stress	Mean parent-child bandwidth (kbps)
AS97	TAG-50	3.460311	13353	537	926.00
	TAG-100	4.318838	11443	293	1229.00
	TAG-200	7.048233	10114	134	1562.00
	ESM	4.022230	13933	390	2407.00
Inet97	TAG-50	4.755937	13904	513	933.00
	TAG-100	9.343069	11585	321	1353.00
	TAG-200	11.080918	11331	290	1559.00
	ESM	7.664771	13091	352	2489.00
AS98	TAG-50	2.574400	17086	343	1034.00
	TAG-100	3.123874	15632	212	1140.00
	TAG-200	5.409245	13268	180	1485.00
	ESM	3.425697	18774	314	2230.00
Inet98	TAG-50	3.574421	14594	283	1279.00
	TAG-100	5.990842	15744	189	1161.00
	TAG-200	10.513578	15611	159	1315.00
	ESM	5.391030	17208	309	2566.00

Table 2 shows the performance of TAG and ESM on AS97, AS98, Inet97, and Inet98. We run three different versions of TAG with respect to $bwthresh$. TAG-50, TAG-100, and TAG-200 denote TAG with $bwthresh = 50$, 100, and 200 kbps, respectively.

TAG-50 gives lower mean RDP than ESM over all configurations. In contrast, the mean RDPs of TAG-100 and TAG-200 are similar to, or even worse than, the mean RDP of ESM. All the TAG configurations exhibit lower mean parent-child bandwidth than ESM (in these simulations, we measure parent-child, not sender to receiver, bottleneck bandwidth). Among the TAG configurations, TAG-200 achieves higher mean bandwidth than TAG-100, which gives higher mean bandwidth than TAG-50. Since TAG only considers bandwidth as a secondary metric, it does not consider bandwidth in its primary tree construction choices. This result shows that $bwthresh$ in TAG must be chosen carefully. A tradeoff between RDP and bandwidth is clearly observed.

In addition, note that fanout of nodes in AS-level topologies is higher than that in router-level topologies. The minus-k path matching algorithm in TAG increases the RDP of 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. TAG with a $bwthresh$ of zero dramatically reduces the RDP (to 1.5 for AS97 and 1.6 for Inet97) at the expense of significant decrease in mean bandwidth.

Total link stress values do not widely vary for TAG-50, TAG-100, TAG-200, and ESM. However, the maximum stress of TAG-50 is higher than the maximum stress of ESM on AS97 and Inet97. The maximum stress of TAG-50 and ESM on AS98 and Inet98 are similar. With a small $bwthresh$ for TAG-50, TAG allows a node to have a large number of children, which results in a high maximum stress. The maximum stress decreases from TAG-50 to TAG-100 to TAG-200. As previously discussed, running the same simulations for TAG with larger $bwthresh$ values reduces the maximum stress.

4.3 Economies of Scale Factor

We compute overlay tree cost via simulations, in order to validate our analytical results from section 3.3. In order to compare results, we assume that one hop used by one point-to-point transfer represents a unit of bandwidth. We therefore add the total stress values for all router-to-router links, and use this quantity to denote tree cost. We run three sets of simulations for unicast, TAG, and IP multicast on the TS2 configuration with 1280 end systems. The complete (not minus-k) path matching TAG is used. This is done to give a fair comparison of simulation results with the analytical results, which modeled complete path matching. The simulation results show that unicast, TAG, and IP multicast cost 16627, 4574, and 1265 respectively. We also plot the normalized overlay tree cost of TAG for a variety of group sizes (using the same methodology as in [13]) in figure 20. The normalized overlay tree cost $L_{TAG}(m)/\hat{u}$ is defined as in section 3.3. The figure shows that $L_{TAG}(m)/\hat{u} \propto m^{0.95}$. The overlay tree cost stabilizes with tree saturation, as with IP multicast. This is consistent with our modeling results.

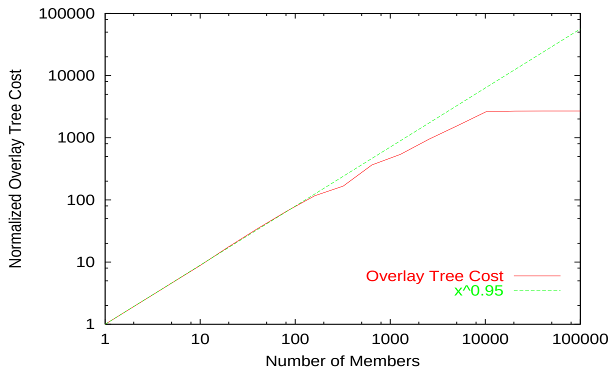


Figure 20: Overlay tree cost versus group size (log-log scale)

5. RELATED WORK

End System Multicast (or Narada) [11, 12] is a clever overlay multicast protocol targeting sparse groups, such as audio and video conferencing groups. End systems in End System Multicast (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. Other application-level multicast architectures include ScatterCast [9], Yoid [18], ALMI [32]. These architectures either optimize delay or optimize bandwidth. In particular, Overcast [24] provides scalable and reliable single-source overlay multicast using bandwidth as a primary metric.

More recently, Content-Addressable Network (CAN)-based multicast [34] 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 employed in [39]. Liebeherr *et al.* investigate Delaunay triangulations for routing in overlay networks in [28]. A prefix-based overlay routing protocol is used in Bayeux [42]. 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 [30].

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 [40]. Overlay networks that detect performance degradation of current routing paths and re-route through other end systems include Detour and RON [3]. Jagannathan and Almeroth [23] 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 heuristic topology-aware application-level multicast protocol called TAG. TAG is single-source or core-based multicast protocol that uses network topology information to construct an overlay network with low delay penalty and a limited number of identical packets. Bandwidth is also considered in tree construction as a secondary metric. TAG, however, works best with high quality underlying routes, and assumes low delay on the last hop(s) to end systems. 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 TAG in building efficient trees for a large number of group members.

We are currently extending TAG to incorporate a tight bandwidth constraint, and delay constrains. With dynamically varying values of the path deviation parameter k and the bandwidth threshold $bwthresh$, a new member can find a better parent, in terms of both latency and bandwidth. We are also considering a hierarchical approach for increasing adaptivity and scalability. This includes using partial topology in a subsequence matching algorithm. We will extend TAG to include other QoS parameters such as power availability in wireless nodes. In addition, we will incorporate TAG into two different applications (a multi-player online game and a video streaming application), and conduct experiments for evaluating the practical aspects and performance of a TAG implementation in the Internet.

7. ACKNOWLEDGMENTS

The authors would like to thank the NOSSDAV 2002 reviewers for their valuable comments that helped improve the paper. This research is sponsored in part by the Purdue Research Foundation, and the Schlumberger Foundation technical merit award.

8. REFERENCES

[1] Performance Measurement Tools Taxonomy. http://www.caida.org/tools/taxonomy/performance.xml.

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

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

[4] P. Aukia, M. Kodialam, P. Koppol, T. Lakshman, H. Sarin, and B. Suter. RATES: A server for MPLS traffic engineering. *IEEE Network*, pages 34–41, March/April 2000.

[5] E. Bacceli and R. Rajan. Monitoring OSPF routing. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[6] 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.

[7] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN Service Availability. In *Proc. of 3rd USITS*, pages 97–108, 2001.

[8] H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Towards Capturing Representative AS-Level Internet Topologies. Technical Report UM-CSE-TR-454-02, Michigan, 2002.

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

[10] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. The Origin of Power Laws in Internet Topologies Revisited. In *Proc. of IEEE INFOCOM*, June 2002.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] A. Feldmann and J. Rexford. IP network configuration for traffic engineering. Technical Report 000526-02, AT&T Labs-Research, May 2000.

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

[19] 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.

[20] R. Govindan and V. Paxson. Estimating Router ICMP Generation Delays. In *Proc. of Passive and Active Measurement*, 2002.

[21] H. Holbrook and B. Cain. Source-Specific Multicast for IP.

Internet-Draft, November 2001.

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

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

[24] 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.

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

[26] C. Labovitz, R. Malan, and F. Jahanian. Internet Routing Instability. *IEEE/ACM Trans. on Networking*, 6(5):515–526, 1998.

[27] 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.

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

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

[30] 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.

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

[32] 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.

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

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

[35] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a Case for Informed Internet Routing and Transport. *IEEE Micro*, 1(19):50–59, January 1999.

[36] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-End Effects of Internet Path Selection. In *Proc. of ACM SIGCOMM*, pages 289–299, September 1999.

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

[38] A. Shaikh, M. Goyal, A. Greenberg, R. Rajan, and K. K. Ramakrishnan. An OSPF Topology Server: Design and Evaluation. *IEEE Journal of Selected Areas in Communications*, 20(4):746–755, May 2002.

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

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

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

[42] 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.