STRESS: Efficient Multicast Shared Trees via Restricted Search

Nian-Feng Tzeng

Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA 70504, U.S.A.

Abstract

A shared tree with a lower end-to-end delay carries out multicast packet delivery more efficiently. This article introduces efficient multicast shared trees via restricted search (STRESS) realized by means of locating the closest on-tree nodes for new group members to join the trees. Such a tree ensures an end-to-end delay shorter than that in a tree built by connecting new members always to a fixed tree node (like CBT or PIM-SM). STRESS is shaped only by multicast group members to yield a tree as small as possible, totally avoiding the difficult task of determining a fixed tree node for member joining. It requires no centralized point to keep track of all on-tree nodes, therefore creating no performance or reliability bottleneck. Two mechanisms for restricted search over multicast trees plus local search are considered and evaluated by simulation. STRESS is shown to be more efficient than CBT, as a result of connecting new members to their nearest known on-tree nodes.

I. Introduction

It is often desirable to let one multicast tree shared by all sources in the group, like PIM (Protocol Independent Multicast) [7] or CBT (Core-Based Trees) [1], for better scalability. Separately, MIP (Multicast Internet Protocol) [13] constructs both group-shared and shortest-path multicast trees, which may co-exist for multicast groups. An IP multicast protocol able to support unicast transparently has been considered [6], facilitating progressive deployment of multicast by supporting unicast clouds. Application-layer multicast has also received attention lately [2, 12].

A single shared tree unfortunately presents several possible problems, as stated earlier [15]. Such a tree often exhibits an unnecessarily long delay for data delivery. To overcome those problems, Guided shared trees (GSTs) have been introduced [15], realized by employing a designated node (dubbed guidance information node, or GIN) for each multicast group to provide helpful information guiding tree growth. While it results in shared trees with improved multicast performance, the earlier proposed GST approach presents a few problems itself. First, information of the whole distribution tree has to be collected and kept in the GIN(s), and this information collection may lead to high control traffic overhead in large networks. Second, tree information maintained in the GIN(s) is often outdated or approximate due to non-negligible propagation delay involved in collecting distribution tree information; this outdated may degrade the quality of GSTs constructed. Third, if only one GIN is used for a distribution tree (to keep traffic overhead low), the GIN suffers from poor scalability and a single point of failure. Fourth, the path chosen by a joining node for connecting the existing tree is the *reverse* shortest path, rather than the shortest path (from the tree to the joining node). Thus, GSTs are usually not best possible for asymmetric networks, often wasting more resources than necessary in Quality of Service (QoS) routing.

This article proposes multicast shared tree construction through restricted search so as to yield efficient information delivery trees without resorting to any single node(s) for gathering all on-tree node information and for serving join requests to provide lists of candidate on-tree nodes. Any multicast group member joins an existing shared tree via restricted search to find the best (i.e., the nearest known) on-tree node for the member to connect through a shortest path (rather than always making a connection to a fixed node, like the core of CBT or the RP of PIM-SM). This proposed work can be realized by means of different search methods, giving rise to various shared trees via restricted search (STRESS) for a given multicast group in a network. Unlike GSTs, STRESS identifies the best on-tree node (as the point of attachment, PA) for a joining member (to connect) through a search over either the distribution tree or the vicinity of the joining member. As the identified node results from a search on-demand, among on-tree nodes themselves, there is no concern of outdated or approximate tree information, nor is any single point of failure present. As those on-tree nodes being searched can estimate their costs toward the joining node, the best on-tree node so chosen offers the shortest path from the tree toward the joining node, as opposed to the reverse shortest path selected under GST construction, especially desirable for asymmetric networks.

The implementation of STRESS depends upon unicast forwarding and requires additional support from the infrastructure. Note that an early QoS-sensitive multicast protocol, QoSMIC [11], calls for both local search and multicast tree search similar to the steps taken by STRESS; however, STRESS avoids the use of any Manager router (needed by QoSMIC) to administer a multicast group (because such a router suffers from exactly the same problems as GSTs do) and employs innovative search procedures over the multicast tree to reduce control traffic overhead, producing needed search results effectively.

This work was support in part by the National Science Foundation under Grant CCR-0105529, by the U.S. Department of Energy under Award Number DE-FG02-04ER46136, and by the Board of Regents, State of Louisiana, under Contract Number DOE/LEQSF(2004-07)-ULL.

II. Pertinent Background

Multicast shared trees

Different protocols for building multicast shared trees had been introduced, noticeably CBT [1], PIM-SM [9] and MIP [13], and they are more favorable, from the standpoint of space overhead (for maintaining tree states at each on-tree nodes) and scalability, than those which construct per-source shortest-path trees. Under the CBT (or PIM-SM) protocol, one node in a domain is selected as the core (or RP), which is known to all routers in the domain, for a multicast group. The choice of the core (or RP) is critical, dictating the shared tree built and thus its data delivery performance. However, selecting a suitable core is hard, since group membership is not fixed and tends to change over time.

Every on-tree router in CBT maintains group-specific state information, namely, a group ID and a list of local interfaces over which every multicast data packet, once received, has to be forwarded (except for the interface from which the packet arrived). A shared tree built according to CBT is not well-shaped, and its multicast packet delivery exhibits a delay dependent on the position of the core.

Guided shared trees

Guided shared trees (GSTs) are developed with an aid of guided information for finding *nearest known* on-tree nodes to connect, upon receiving group join requests [15]. Such information is available at a designated node, called GIN (guidance information node), which keeps track of on-tree nodes, as the delivery tree grows. It helps to shorten end-to-end delay over a multicast shared tree so developed. When a new joining node arises, the GIN provides a list of candidate on-tree nodes which are determined, based on location indicators (LIs), to be closest to the joining node. The joining node, upon receiving those candidate on-tree nodes, chooses the best on-tree node to connect, according to its kept unicast routing information. LIs help to shape shared tree expansion [15], yielding an efficient delivery tree superior to that resulting from joins always to a fixed point (like the core of CBT or the RP of PIM-SM).

III. STRESS Outline

This section outlines how to construct our shared trees via restricted search (STRESS) for efficient multicast within a domain (i.e., an administratively-independent network). Our approach lies in identifying a small number of candidate on-tree nodes via restricted search for a joining node so that the best point of attachment (among those candidate on-tree nodes) can be selected, appropriately shaping tree growth as subsequent group members join. It does away with the core (or RP) of CBT (or PIM-SM) such that a multicast tree so constructed is no longer rooted at any single node, but is shaped according to the locations of group members in a way that produces a nearly "minimal" possible tree to cover them. STRESS relieves totally the need to choose one proper node for a multicast group (since the node critically affects the topology of a multicast tree), while avoiding the problems caused by GSTs proposed earlier.

Group joins

Hosts join multicast groups through their respective first-hop routers following the IGMP protocol [3]. When such a router receives its first multicast join request for a given group, it invokes the STRESS protocol to join the multicast tree of the group. If the tree has not existed yet (signifying that the join request is the very first one in the domain and thus a multicast group is to be created), the router informs all multicast routers in the domain of the creation and existence of the group (specified by its group ID). The first node (i.e., router) that starts a multicast group serves as the tentative point of attachment (TPA) for subsequent node joins. Unlike the core (or RP) of CBT (or PIM-SM), the TPA of STRSS has no impact on the eventual tree topology nor the performance of multicast delivery. In fact, STRESS allows an arbitrary number of on-tree nodes to announce themselves as TPAs for a given group, and it is up to every future node (i.e., a multicast router) to (register and) decide its preferred TPA (e.g., according to distance, cost, delay, or other measures) when it is to join the tree.

It should be noted that STRESS works even when a fixed node (or a small number of fixed nodes) is designated as TPA(s) for all possible groups in a domain, much like the core (or candidate core list) for CBT. In this situation, the fixed node (or the element indexed by the group ID among a small number of fixed nodes) is treated as the very first member of a multicast group, serving as the initial TPA.

STRESS calls for search to locate the best possible on-tree node for any new node to connect. This search is conducted from two prospects: one is over the multicast tree (called *tree search*) and the other is over the vicinity of the joining node (called *local search*). It is described below.

1. Tree search is initiated by the joining node, which contacts its TPA by sending a join request. As soon as the join request hits the first on-tree node or the TPA, tree search starts there to select a few candidate points of attachment (PAs). Candidate PAs are forwarded to the joining node for selection. Search over a multicast tree is restricted to only promising branches (rather than the whole tree). The next section will provide different mechanisms for restricted search to select candidate PAs and for informing the joining node of the search results.
2. Local search has been suggested to find multiple paths satisfying the QoS requirement(s) in the neighborhood of a new group member [4, 11]. Our local search follows a similar approach, making use of reverse path forwarding to limit flooding traffic. However, the search scope is restricted by a threshold decided according to the distance between a joining node and the multicast tree. It aims to locate promising on-tree nodes as candidate PAs for the joining node to select.

Obviously, tree search and local search may be carried out in parallel or in sequence. Search in parallel is desirable to set a small threshold for local search and tends to lower the joining latency, whereas search in sequence may reduce traffic overhead and could yield better search results, if the threshold of local search is set to be d–1, where d is the

distance between the joining node and the TPA (or the first on-tree node along the path between them).

The joining node then chooses the *nearest* PA among those candidate PAs received during search, referred to as the permanent PA (PPA). One more join request is originated from the joining node toward PPA to establish a connection (by setting up an appropriate multicast routing table entry at each node along the connection) from the node to the multicast tree, finishing the join process.

Two-phased join strategy

STRESS adopts a two-phased join strategy in order to lower the time period a joining node has to wait before starting to receive multicast packets delivered over the tree. Specifically, when tree search is initiated by the joining node using a join request toward its TPA, the request may also serve to establish a *tentative connection* from the node to the tree, referred to as the Phase-I join. As soon as the join request meets the first on-tree node or the TPA, the Phase-I join is finished and multicast data can be delivered immediately to the new node without waiting for it to join the tree through a connection to the PPA, called the Phase-II join (which occurs after search has completed and the PPA has been chosen from those candidate PAs).

Under this two-phased join strategy, the join latency is reduced significantly at the expense of increased control traffic overhead. As the Phase-I join creates only a tentative path to the new node, such a path may be maintained using either a soft-state or a static entry in the multicast routing table for the group at each involved node along the path. A soft-state entry remains, as long as the new node keeps sending join refreshes periodically before the entry times out. The periodic refreshes contribute to additional traffic overhead, and the refresh frequency dictates the overhead degree. Once it has chosen its PPA and issued a permanent join request, the new node ceases to generate join refreshes over the tentative path, effectively removing that path. On the other hand, if a static entry is employed to maintain the tentative path, a prune request is issued by the new node to tear down the path after a permanent join request has been sent. This prune request contributes to extra traffic overhead.

IV. STRESS Details

This section describes in detail restricted search to realize STRESS, followed by some discussion on the two-phased join strategy. STRESS makes use of different control messages: TNT_JOIN (for a tentative join), PROBE (for probing a path), BID (for an on-tree node to inform the joining node), PMN_JOIN (for a permanent join), PRUNE (for removing a tentative path).

Restricted search

Search is conducted when a router (i.e., node) receives a request from any of its hosts for joining a multicast group (through IGMP [3]), to which the router does not belong yet. It intends to locate the closest on-tree node as the PPA for a new node in order to establish a shortest path from the multicast tree to the new node. Tree search is governed by different mechanisms in order to restrict the search scope over a multicast tree, aiming to reduce control traffic overhead without compromising the quality of search results.

When a node (N_j) intends to join a multicast tree, it sends a TNT_JOIN message to its chosen TPA. The message is forwarded in accordance with unicast routing information kept in the intermediate nodes until it hits either an on-tree node or the TPA, called N_{th}. Restricted search takes the following steps:

1. Search over the tree starts from N_{th}, along active interface I_{ai} (with respect to the multicast group) of N_{th} or of any node visited (by a search probe), provided (i) the clue or decision associated with I_{ai} calls for a PROBE message to be sent over it and (ii) the PROBE message does not arrive from I_{ai}. A PROBE carries with it (i) ID of the on-tree node, say N_Ψ, which is closest to N_j among those visited by the PROBE so far, and (ii) the distance from the on-tree node to N_j, say Ψ (which can be told directly from the unicast routing table kept in the node).
2. When a PROBE proceeds to the next on-tree node, Ψ is employed to decide if a closer node is encountered. Initially, N_Ψ and Ψ are set to N_{th} and the distance from N_{th} to N_j, respectively.
3. Step 1 repeats at the visited node, unless no further search can be performed (i.e., no PROBE message is issued) at the node. Such a node then sends one BID message to N_j, provided that its arrival PROBE carries $N_\Psi \neq N_{th}$.

Each BID message carries N_Ψ and Ψ, which refer to the nearest on-tree node and its distance to N_j (along the shortest path) discovered by a PROBE. N_j chooses the "overall" nearest on-tree node, denoted as PPA (permanent PA), according to BID messages collected there. A PMN_JOIN message is then sent to PPA for creating a permanent path between the tree and N_j.

This search procedure relies entirely on the clue or decision associated with every active interface probed to restrict its search scope over a multicast tree, as outlined in Step 1 above. Two mechanisms are considered for our restricted search, one maintaining a clue for each active interface (registering the "location range" of the sub-tree rooted at the interface) while the other following a ticket-based decision. They are described next in sequence.

Clue-based tree search. The clues of active interfaces can be obtained after a location indicator (LI) is assigned to each node in a domain. Under ideal assignments, two nodes are close to each other if their respective LIs differ by a small amount; the closer the two nodes, the smaller gap between their LIs. All nodes in a domain are involved in establishing LIs (before any multicast starts), and the same LI assignment holds for every group. The assignment remains unchanged until the network gets modified by adding or dropping nodes/links.

Initially, contiguous integer numbers are assigned to those nodes which are in vicinity [15], as their LIs. In Fig. 1, for example, the LI assignment starts from node C (the core), with its three immediate neighbors (nodes B, A, and

F) assigned with LI = 2, 3, and 4, respectively. The neighbors of B (whose LI is the smallest after C's), if any, are examined and assigned with the next numbers, if they are without LI yet. Node D is thus assigned with LI = 5. Next, F's neighbors are checked and assigned with appropriate LIs, e.g., G, E, and I with LI = 6, 7, and 8, respectively. This assignment process repeats until all nodes get their LIs. A procedure is given in [15] to handle this assignment. After an initial assignment of LIs to nodes, LI re-computation is carried out by taking the weighted average of LIs of a node and all its immediate neighbors to enhance the quality of the LI assignment. This often takes just a few iterations (say, 3) to reach fairly good assignments, even for a large domain (say, composed of 150 nodes as shown in [15]).

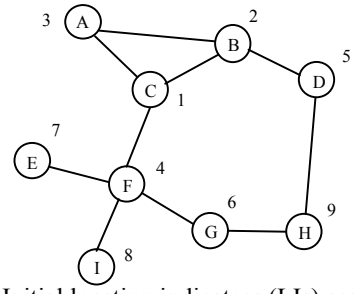

Fig. 1. Initial location indicators (LIs) assigned to nodes.

The LIs assigned to nodes in a domain serve as the basis for producing clues for restricted search. A clue is associated with each active interface and is in the form of the LI range of nodes of the sub-tree rooted at the interface, namely, signifying the largest and the smallest LI values of all nodes in the sub-tree. In Fig. 2, for example, clues for active interfaces are given inside pairs of brackets. At node C, its interfaces #1 and #2 have clues of <14.2, 14.2> and <12.4, 12.4>, respectively, where the former (or latter) clue signifies the LI range of the node(s) in the sub-tree rooted at interface #1 (or #2). Notice that interface #3 of node C has a clue of <1, ∞>, indicating that tree search always takes that branch; more details about this kind of interfaces will be given next. The clue associated with interface #1 of node E is <11.0, 14.2>, which reflects the LI range of nodes of the sub-tree rooted at the interface. Clues for a tree are kept in the multicast routing table, and those clues for a group are fetched at the same time when the table entry of the group is accessed. Tree search involves no extra table lookup time.

As clues are accumulated along the tree after a new node is joined, TPA automatically becomes a specific node, called the *anchor*, as follows. After a new node joins a tree successfully, the LIs of those nodes on the established path (connecting the new node and the tree), in a form known as the *LI span*, is forwarded along the interface with clue <1, ∞> at each tree node. Note that there is exactly one interface with clue <1, ∞> at each node except the tree anchor, which *does not have* such an interface. Forwarding the LI span thus stops at the anchor. The interface with clue <1, ∞> leads to the tree anchor, and such an interface is always taken during tree search. On its way to the anchor, the LI span is employed to modify the clue of every arrival interface (through which it arrives at a tree node). In Fig. 2, for example, node D is joined by a path through node G to reach the tree at node F. The LI span of nodes along the established path is (15.7, 16.1), which serves as the clue of interface #2 of node F, <15.7, 16.1>, and which also causes the clue of interface #1 of node C to be modified when it is forwarded to the anchor. Likewise, the clues of arrival interfaces of nodes E and A are updated accordingly, where A is the tree anchor. Meanwhile, node D has only one active interface, which is initialized with <1, ∞>, as the interface leads to the tree anchor. The resulting clues at all tree nodes after the LI span of the newly established path has reached the anchor are shown inside pairs of parentheses in Fig. 2. Clues of active interfaces are accumulated as nodes are added to the tree, and clue accumulation is carried out in a distributed manner over the tree automatically without the involvement of any single point.

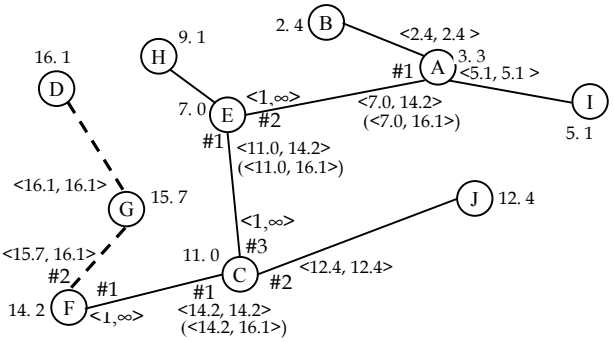

Fig. 2. Shared tree of eight nodes with clues for certain interfaces shown (and others left out for clarity). Clues affected by the D join are given inside parenthesis pairs.

Note that a tree anchor is fundamentally different from the core (or RP) of a multicast tree developed under CBT (PIM-SM) in three aspects. First, a tree anchor is for the use of gathering clues in support of restricted search, and it has nothing to do with how a multicast tree is shaped and performs. Second, no router (whether on the tree or not) in the domain needs to know where the anchor is, since the LI span of any new branch added is forwarded in a distributed manner toward the anchor automatically. Third, a tree anchor does not have to be fixed during a multicast session; a different on-tree node can be selected at any time as a new anchor, with clues associated with active interfaces of nodes along the path between the current anchor and the new anchor reestablished. Once a new anchor is chosen, it contacts the current anchor (by following active interfaces with clue <1, ∞>). Clue reestablishment is performed in a distributed manner and starts from the current anchor toward the new one, with two interfaces in each intermediate node reestablishing their clues accordingly. In Fig. 2, as an instance, if node C is selected as a new anchor, three nodes, A, E, C, are involved in clue reestablishment. Specifically, interface #1 of node A, leading to the new anchor, now gets a clue of <1, ∞>, interfaces #2 and #1 of (intermediate)

node E have clues of <2.4, 5.1> and <1, ∞>, respectively, while interface #3 of node C possesses a clue of <2.4, 9.1>.

Step 1 of restricted search outlined earlier is now stated as follows. A PROBE message is sent over I_{ai}, if LI of node N_j, denoted by $LI(N_j)$, satisfies $C\_L - \Delta \leq LI(N_j) \leq C\_H + \Delta$, where Δ is a constant while C_L and C_H are the lower and the upper bounds of the LI range maintained for I_{ai}. Introducing Δ in the decision expression aims to compensate for the fact that LIs do not perfectly reflect location proximities of nodes, by allowing search over neighborhoods of tree nodes revealed by a clue. If a sub-tree is indicated by its clue to contain nodes all far away from N_j, search into the sub-tree likely is wasteful and should be avoided, restricting the search scope. It will be shown using simulation that clues can lower search traffic overhead substantially with little sacrifice to the quality of search results.

Ticket-based tree search. Ticket-based routing was considered earlier for identifying paths which meet specified QoS requirements for a given source and destination pair [5, 16]. Its basic idea is to let the source issue a number of tickets according to the QoS requirements, where each ticket is a permission to probe one path. Each probe is required to carry at least one ticket. At an intermediate node, a probe with more than one ticket may be split into multiple ones, in search of different branches from the node. Different tickets are assigned to those split probes, depending on the distances from the node along separate branches to their destination. The number of paths probed is bounded by the total number of tickets issued initially. Ticket-based search has never been considered previously for non-QoS applications.

Adopting ticket-based search in a multicast tree to identify the nearest PA for a new node to join, we have to make two modifications. First, the number of tickets is not issued by N_j (the new node), but by N_{th} (the first on-tree node encountered by the TNT_JOIN message originated from N_j), according to the distance from N_{th} to N_j, say Ψ. Specifically, the total number of tickets produced initially is given by a function of Ψ and ε, where ε is a small constant larger than 1, say 1.5, which dictates search traffic overhead:

$$\text{ceiling}\ (\varepsilon \times \Psi). \qquad (1)$$

This expression signifies that if N_j is far away from the tree, more tree paths have to be probed for a better chance of finding the best PA present. In practice, the initial ticket number is selected to be no more than a threshold (say, 6). Conversely, if N_j is close to the tree, N_{th} may be a good PA already, making it unnecessary to explore the tree widely. This guideline keeps search traffic overhead low adaptively while ensuring good quality of the tree built, for a given ε. Second, when a PROBE arrives at a tree node with more than two active interfaces, the tickets carried by the PROBE are distributed *as evenly as possible* to those forwarded PROBEs over the interfaces other than the one where the PROBE arrived, such that each forwarded PROBE carries at least one ticket. This is different from earlier ticket-based QoS routing, where tickets are split unevenly for forwarded PROBEs [5]. (Note also that our search is limited to the multicast tree, whereas earlier ticket-based routing exploits *all interfaces* of every node visited.) If there are more active interfaces than the number of tickets carried by an arrival PROBE at a node, certain interfaces of the node will not be taken and those interfaces are *randomly selected*. This second modification realizes the probing decision (on each active interface) called upon in Step 1 of restricted search.

It is worth noting that ticket-based tree search described above follows greedy expansion by splitting tickets into multiple PROBEs at earliest on-tree nodes encountered, irrespective of whether such nodes are closer to N_j than N_{th} or not. Alternatively, one may adopt a less greedy approach by splitting tickets only when a newly visited node is closer to N_j than its predecessor. This approach intends to further restrict the search scope by not exploring multiple paths along a branch which seems unpromising. Naturally, it may lead to somewhat lower STRESS quality. According to Step 3 of restricted search given above, the BID message of each path is sent to N_j only when the path cannot be explored further. It is possible to send a BID message earlier (and stop path exploration), when a PROBE message fails to find a closer node after traversing k nodes consecutively. This early BID report cuts probe traffic overhead by terminating a path search sooner whenever it is deemed unpromising. Apparently, a smaller k leads to more traffic overhead reduction at the expense of a higher chance of missing the absolutely nearest PA. The study of trade-offs between k and STRESS quality is not included in this article.

Local search

Our local search follows the two-phased join strategy, issuing a TNT_JOIN message from a new node (N_j) first. It starts only *after* a *reply* to the message has reached N_j, and the reply brings back the ID of N_{th} (i.e., the first on-tree node encountered by the message). The distance from N_j to N_{th}, say Ψ', is used for determining the radius of flooding (i.e., the initial TTL of a probe message), as follows: if $\Psi' \leq \zeta$, the initial TTL is set to $\Psi'-1$; otherwise, it is set to ζ, where ζ is a threshold which governs search traffic overhead. Upon receiving a PROBE, a node broadcasts it in accordance with reverse path forwarding to limit traffic. When a PROBE encounters an on-tree node, the PROBE is suspended there and a BID message is delivered to N_j. If N_j receives no BID message, its PPA is N_{th} and the tentative connection becomes permanent; otherwise, its PPA is chosen from all BID messages received. In an asymmetric network, a BID could pass through an on-tree node explored by another PROBE. In that case, the BID is suppressed (to reduce traffic overhead) since an earlier BID (which is a better one) has been produced by the explored node. Such a case does not exist in symmetric networks.

If ζ is set to be the network radius, an absolute nearest on-tree node is guaranteed to be revealed. However, this comes with a prohibitively high traffic overhead, expressed in the worst case by

$$\gamma \times (\gamma - 1)^{\theta-1}, \qquad (2)$$

where γ and θ are the average degree of nodes and the network radius, respectively. This expression denotes flooding over the whole network and is often much larger than the average multicast tree size, say χ. If local search

overhead is restricted to be no more than that of tree search (which is bounded by χ), we have a threshold (ζ) of

$$\zeta \leq \ ln(\chi/\gamma) / ln(\gamma-1) + 1. \qquad (3)$$

If tree search and local search are performed in parallel, Eq. (3) can be used for choosing an appropriate threshold. If they are conducted in sequence (to improve the quality of search results), a larger threshold is desired and chosen.

Discussion on two-phased joins

The TNT_JOIN message serves to build a tentative path between N_j and N_{th} for multicast packets to flow from N_{th} to N_j sooner. Such a path is to be torn down by a PRUNE message issued from N_j at an appropriate time after the PMN_JOIN message has been sent. The object is to avoid losing any multicast packets due to removing a tentative path prematurely before a permanent path has commenced packet delivery. Therefore, a tentative path is torn down after the permanent path has delivered a duplicate packet to N_j. Note that there is *no concern* about the existence of a loop, since N_j stops forwarding any multicast packet delivered over the tentative path. No additional buffer is required for packets delivered over the tentative path (other than what is necessary to implement a sliding window or selective repeat protocol), in order to properly discard those duplicate packets delivered over the permanent path.

After a tentative path is established (by N_j), it is possible that a separate new node, say N_j', is to join the tree through the same TPA. If the tentative path created from N_j' shares a part of the tentative path established by N_j, a special mark has to be kept at the node where these two tentative paths meet, say N_m. The mark is to prevent a later PRUNE message (issued by N_j or N_j') from tearing down the shared path segment prematurely. When N_m receives a PRUNE message, its mark is removed, and the PRUNE message is suspended there. A subsequent PRUNE message, if any, will make that shared path segment be removed. In general, a counter (rather than a mark) may be employed to deal with any number of tentative paths sharing a path segment; when a TNT_JOIN message reaches N_m, its counter (which is initialized with 0) is increased by 1. The counter is decremented by 1 upon receiving a PRUNE message, which is then dropped there if the counter has not reached 0 yet.

V. Performance Evaluation

Performance of STRESS under different network sizes is evaluated using *ns*-2 [10]. The simulation results are compared with those of CBT, in terms of two performance metrics: mean delay and number of on-tree nodes. The first metric reflects the average time taken to multicast a packet from its source to all members. A more efficient delivery tree enjoys a lower mean delay for the same multicast group. The second measure registers how many nodes (routers) are involved in the shared tree built for a multicast group. Each on-tree node maintains tree states, and thus it is desirable to keep as few such nodes as possible for a given group. As STRESS search mechanisms involve varying control traffic overhead amounts, the overhead gauge of a search (in terms of hops, where a hop refers to a control message moving from one node to its immediate neighbor) is collected.

Evaluation setup

Different network topologies for various sizes have been generated following the tiers network generation model [8] for evaluation. We used only the lowest tier to capture the domain topology, with a random distribution for node degrees [14] ranging from 1 to 6. For each network with size *n*, we assigned up to *n*/5 network nodes randomly as sources, each generating multicast data packets with size 1K bits at the constant rate of 1 packet per 9 *ms*. It is observed that the packet generation rate has little impact on performance metrics of interest. Multiple group joining sequences were produced for evaluating each network topology and each network size. A sequence finishes when *n*/5 network nodes have joined a group.

Every multicast packet is time-stamped when generated by a source in order to measure its delay (in *ms*) to reach each group member. Such a delay measured at each member is accumulated to get the mean delay. We simulated a range of network sizes (*n*) and found that the results follow a similar trend for various *n*. Therefore, data values presented in subsequent figures are for $n = 160$ only, and they are the average results over 2 different topologies each under 12 joining sequences. The simulation results illustrate multicast trees STRESS built under different search mechanisms, with TPAs kept unchanged throughout multicast sessions.

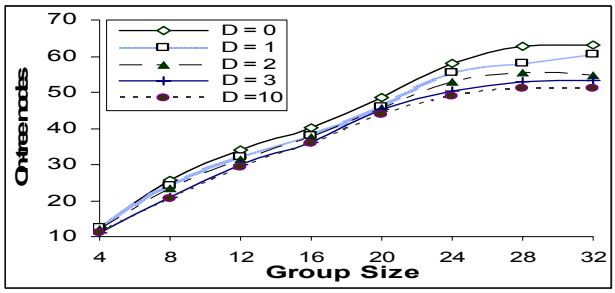

Fig. 3. On-tree nodes versus group size under various D.

Simulation results

To assess the impact of D (i.e., an indicator for searched neighborhoods of tree nodes revealed by a clue) on quality of the trees built, we varied D in our simulation under different group sizes, with results depicted in Fig. 3. For a given group size, the number of on-tree nodes drops as D grows, reflecting a better multicast tree constructed. This is expected since a bigger D permits clue-based search to exploit larger neighborhoods. The benefits diminish swiftly as D increases and become negligible when D exceeds 3. Because a bigger D exhibits higher traffic overhead, it appears that D equal to 3 is adequate and should be adopted.

In Fig. 4, we illustrate performance results of STRESS for the network size of 160, where the number of group members varies from 4 to 32. The CLU bars refer to the results of our clue-based search with D = 3, the TIC bars

denote those of ticket-based search with ε = 1.5 (for Eq. (1) chosen to be upper bounded by 5), and the LOC bars are results of local search with ζ = 4, whereas ODT (optimum distribution tree) bars indicate those of "best" trees possible to be built by exploiting the whole tree (say, resorting to unrestricted ticket-based search with the number of tickets initially set to ∞) in response to each joining request. The CBT results are included as well, and they are clearly worse than all STRESS ones in terms of the two performance metrics gathered. When there are 20 group members, for example, CBT experiences mean delay of 0.128 *ms*, in contrast to 0.105 *ms* of CLU, translating to a 18% reduction. Even under restricted search, CLU yields the results close to those of ODT (via unrestricted search), with significant overhead reduction, as illustrated in Fig. 5.

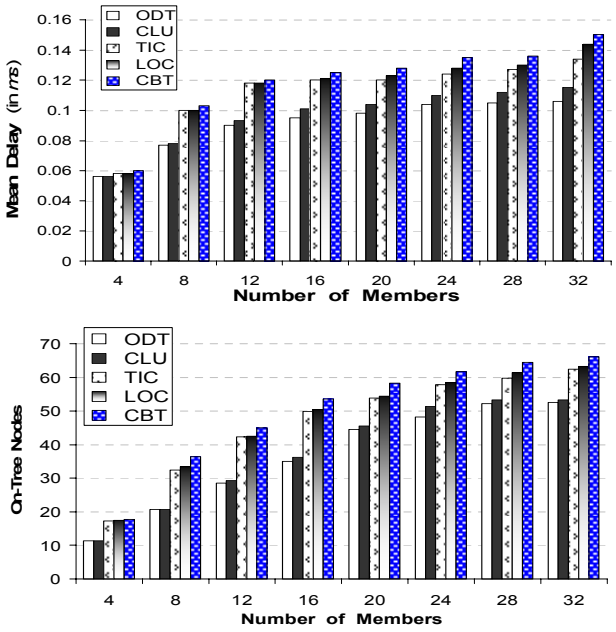

Fig. 4. Performance results vs. group size for n = 160.

Cumulative traffic overhead (due to joins of all group members combined) as a function of the group size for different STRESS mechanisms (whose parameters are the same as those given in Fig. 4) is depicted in Fig. 5, where the CBT results are also included. Since CBT involves no search at all, its overhead is lowest. With restricted search making use of clues, CLU exhibits lowest overhead values among all STRESS mechanisms, making it most desirable.

VI. Conclusion

Efficient multicast shared trees built via restricted search have been introduced to discover nearest on-tree nodes in response to joining requests of new group members. A tree so constructed, known as STRESS, enjoys smaller mean latency for multicast packet delivery due to a lower end-to-end delay. STRESS requires no centralized point to keep track of all on-tree nodes, thus preventing any performance or reliability bottleneck from occurring. Restricted search over the tree results from the use of either clues to explore only promising branches or tickets to limit the search scope. Local search is considered as well. Simulation outcomes have revealed that STRESS indeed is more efficient (in terms of performance measures of interest examined) than its CBT counterpart, with clue-based tree search being most advantageous.

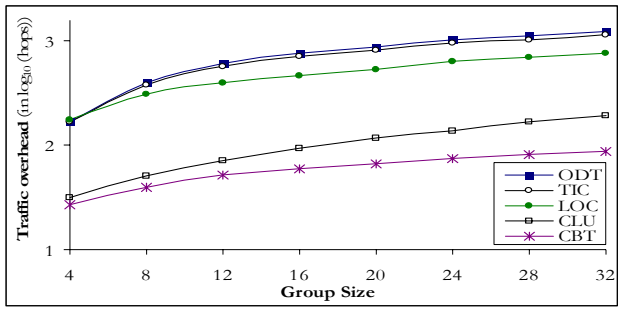

Fig. 5. Cumulative traffic overhead versus group size.

References

[1] A. Ballardie, P. Francis, and J. Crowcroft, "Core-Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing," *Proc. ACM SIGCOMM'93*, Sept. 1993, pp. 85-95.

[2] S. Banerjee *at al.*, "Scalable Application Layer Multicast," *Proc. ACM SIGCOMM'02*, Aug. 2002.

[3] B. Cain *et al.*, "Internet Group Management Protocol, Version 3," Internet RFC 3376, Oct. 2002.

[4] K. Carlberg and J. Crowcroft, "Building Shared Trees Using a One-to-Many Joining Mechanism," *ACM Computer Communication Review*, pp. 5-11, Jan. 1997.

[5] S. Chen and K. Nahrstedt, "Distributed Quality-of-Service Routing in Ad-Hoc Networks, *IEEE J. on Selected Areas in Communications*, vol. 17, pp. 1488-1505, Aug. 1999.

[6] L. Costa *et al.*, "Hop By Hop Multicast Routing Protocol," *Proc. ACM SIGCOMM'01*, Aug. 2001, pp. 249-259.

[7] S. E. Deering *et al.*, "The PIM Architecture for Wide-Area Multicast Routing," *IEEE/ACM Trans. on Networking*, vol. 4, pp. 153-162, Apr. 1996.

[8] M. Doar, "A Better Model for Generating Test Networks," *Proc. IEEE Global Telecommunications Conf.*, Nov. 1996.

[9] D. E. Estrin *et al.*, "Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification," Internet RFC 2362, June 1998.

[10] K. Fall and K. Varadhan, *The ns Manual*. UC Berkeley, LBL, and USC/ISI, 2001. http://www.isi.edu/nsnam/ns/.

[11] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of Service Sensitive Multicast Internet Protocol," *Proc. ACM SIGCOMM '98*, Sept. 1998, pp. 144-153.

[12] J. Jannotti *et al.*, "Overcast: Reliable Multicasting with an Overlay Network," *Proc. 4th Symp. on Operating Systems Design and Implementation*, Oct. 2000.

[13] M. Parsa and J. Garcia-Luna-Aceves, "A Protocol for Scalable Loop-Free Multicast Routing," *IEEE J. on Selected Areas in Comm.*, vol. 15, pp. 316-331, Apr. 1997.

[14] H. Tangmunarunkit *et al.*, "Network Topology Generators: Degree-Based vs. Structural," *Proc. ACM SIGCOMM '02*, Aug. 2002.

[15] Nian-Feng Tzeng and P. Alla, "Guided Shared Trees for Efficient Multicast in Large Networks," *Proc. 2003 IEEE Int'l Conf. on Communications*, May 2003.

[16] L. Xiao *et al.*, "The Enhanced Ticket-Based Routing Algorithm," *Proc. 2002 IEEE Int'l Conf. on Communications*, Apr./May 2002.