Distributed Delay Constrained Multicast Routing Algorithm with Efficient Fault Recovery

Hasan Ural and Keqin Zhu
School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, KIN 6N5 Canada

Existing distributed delay constrained multicast routing algorithms construct a multicast tree in a sequential fashion and need to be restarted when failures occur during the multicast tree construction phase or during an on-going multicast session. This article proposes an efficient distributed delay constrained multicast routing algorithm that constructs a multicast tree in a concurrent fashion by taking advantage of the concurrency in the underlying distributed computation. The proposed algorithm has a message complexity of O(*mn*) and time complexity of O(*n*) in the worst case, where *m* is the number of destinations and *n* is the number of nodes in the network. It constructs multicast trees with the same tree costs as the ones constructed by well-known algorithms such as DKPP and DSHP while utilizing 409 to 1734 times fewer messages and 56 to 364 times less time than these algorithms under comparable success rate ratios. The proposed algorithm has been augmented with a fault recovery mechanism that efficiently constructs a multicast tree when failures occur during the tree construction phase and recovers from any failure in the multicast tree during an on-going multicast session without interrupting the running traffic on the unaffected portion of the tree. © 2005 Wiley Periodicals, Inc. NETWORKS, Vol. 47(1), 37–51 2006

Keywords: distributed multicast routing; constrained Steiner tree; delay constrained multicast tree; fault recovery

1. INTRODUCTION

An important part of the functionality of routing is to select a route along which sufficient resources should be reserved to meet various requirements specified by applications, such as network bandwidth and maximum message delay [5]. This functionality is fundamental for guaranteeing real-time data to be delivered to destinations with an acceptable delay. Multicast routing builds a routing tree (called *multicast tree*), which is rooted from a source node s and contains all nodes in a set S of m destination nodes in a network of n nodes. The reasons for forming multicast routes on trees stem from the facts that data can be transmitted simultaneously to various destinations along the branches of the multicast tree; and the number of copies of the transmitted data can be minimized by replicating the data only at the forks in the multicast tree.

Algorithms for constructing multicast trees take into consideration two important factors [23, 28]. The first factor is the constraint (placed by the application performing the multicast) on the end-to-end delay along the individual paths from the source to each destination. The second factor is the requirement for minimizing the cost (e.g., network bandwidth utilization) of the multicast tree that is the sum of the costs on the edges in the multicast tree. The minimum cost tree is called a *Steriner tree*, and the problem of finding a Steiner tree is known to be NP-complete [15, 20]. The delay constrained minimum cost tree is called a *constrained Steiner tree* [17]. The problem of finding a constrained Steiner tree is also NP-complete [17].

Various algorithms based on heuristics for constructing constrained Steiner trees (i.e., for solving the source-initiated, delay constrained multicast routing problem) have been developed in recent years. In general, these algorithms are classified [4] as *source-based* (or *centralized*) multicast routing algorithms such as KPP [16, 17], CDKS [33], CKMB [34], CAO [38], DCMA [39], and BSMA [40] or *distributed* multicast routing algorithms such as DKPP [18, 19] and DSPH [14]. Because the distributed multicast routing algorithms do not require the entire network status information maintained in the source node, they elude the scalability problem encountered by the centralized multicast routing algorithms in very large networks.

However, there are several problems with the existing distributed delay constrained multicast routing algorithms that may be classified as minimum spanning tree (MST) based or shortest path (SP) based. An MST-based algorithm, like DKPP [19], is a distributed MST algorithm as described in [13], which mimics Prim's MST algorithm [9, 24] and runs with a high complexity of $O(n^3)$ in the number of messages and time. Also, an MST-based algorithm has a very low success rate (which is the number of trials to build a multicast

Received December 2004; accepted September 2005
Correspondence to: H. Ural; e-mail: ural@site.uottawa.ca
Contract grant sponsor: Natural Sciences and Engineering Research Council of Canada; contract grant number: RGPIN 976.
DOI 10.1002/net.20090
Published online in Wiley InterScience (www.interscience.wiley.com).

tree successfully divided by the number of total trials) in constructing a delay constrained multicast tree especially under tight delay constraints. On the other hand, an SP based algorithm, like DSPH [14], is less costly in terms of message and time complexity [i.e., O(mn) each] than DKPP, but it has a fatal deficiency; that is, the algorithm will not be able to find a solution if the delay constraint is less than the maximum delay of a cost-based shortest path tree. It is obvious that a delay constrained multicast tree exists as long as the delay constraint is larger than the maximum delay of a delay-based shortest path tree. This means that DSPH may not solve the delay constrained multicast tree problem for a large number of cases where a delay-constrained multicast tree exists.

A common problem with these algorithms is that they do not take into account the changes in the topology of the network (e.g., node failures) [4]. When there is a change in the network topology during the construction of a multicast tree, these algorithms fail to complete the construction of the multicast tree and they have to be restarted. Hence, these algorithms will likely have a low success rate for the construction of a multicast tree. Moreover, when there is a change in the network topology after the multicast tree is built and during an on-going multicast session, the application performing the multicast has to restart the algorithm to rebuild the multicast tree for the changed network topology, which interrupts the running traffic for all members in the multicast session. Restarting the process of building a multicast tree could be considered as a fault recovery approach, called *naive fault recovery approach* [35].

In this article, we propose an efficient SP-based distributed delay constrained multicast routing algorithm that builds the multicast tree in a concurrent fashion. The proposed algorithm has a very low message and time complexity of O(mn) and O(n), respectively, and has a near-zero failure rate in constructing a multicast tree even under very tight delay constraints without any limitation on the value of delay constraints. The simulation results reported in this article show that the proposed algorithm significantly outperforms the existing distributed delay constrained multicast routing algorithms, which construct the multicast tree in a sequential fashion and thus do not take advantage of the concurrency in the underlying distributed computation. We use the same comparison method as used in DKPP [19] and DSPH [14], among many others; that is, comparing the proposed algorithm with the best algorithm in its class instead of the optimal solution. As indicated in [19], for large networks, finding optimal solutions is impractical.

We augment the proposed algorithm with a fault recovery approach that considers node failures in a network and recovers from these failures. Although we only consider the node failures in the proposed algorithm, the approach can easily be extended to cover other types of changes in the topology of the network such as link failures and to cover the dynamic membership. The proposed fault recovery approach constructs a multicast tree when failures occur during the construction of a multicast tree and recovers from any failure in the multicast tree during an on-going multicast session. The fault recovery actions taken by the proposed approach during the construction of a multicast tree and during an on-going multicast session are transparent to applications initiating and performing the multicast. Moreover, the fault recovery actions taken during an on-going multicast session reconnect the disconnected subtree of a multicast tree without interrupting the running traffic on the unaffected portion of the tree. The results of the analysis and simulation studies reported in this article show that the proposed fault recovery approach not only is effective, but also is efficicnt, which gives better performance in terms of the number of exchanged messages and convergence time than the naïve fault recovery approach.

The rest of this article is organized as follows. Section 2 gives the problem definition and presents the proposed algorithm. Section 3 introduces the augmentation of the proposed algorithm with an effective and efficient fault recovery approach. Section 4 discusses the performance of the proposed algorithm and its augmentation. Section 5 reviews the related work and Section 6 gives the concluding remarks.

2. THE PROPOSED ALGORITHM

2.1. Preliminaries

Before we present the proposed algorithm, we define the related terminology by using the notation introduced in [35]. Formally, the constrained Steiner tree can be defined as follows: consider a network modeled as a connected, directed graph $G = (V, E)$, where nodes in node set V represent network nodes and edges in edge set E represent links. In addition, there are two values associated with each link $e (e \in E)$: delay $D(e)$ and cost $C(e)$. The link delay $D(e)$ is the delay a packet experiences on the corresponding link, the link cost $C(e)$ is a measure of the utilization of the corresponding link's resources. Links are asymmetrical, namely the cost and delay for the link $e = (i, j)$ and the link $e' = (j, i)$ may not be the same.

Then, given a network $G = (V, E)$ with n nodes, a source node $s (s \in V)$, a set of m destination nodes $S (S \subseteq V - \{s\})$ called *multicast group*, and a delay constraint Δ, a tree $T (T \subseteq G)$ rooted at s that spans S such that (i) for each node v in S, the delay on the path from s to v is bounded above by Δ; that is $\Sigma_{e \in P(s,v)} D(e) < \Delta$, where $P(s, v)$ is the unique path in T from s to v; and (ii) the tree cost $\Sigma_{e \in T} C(e)$ is minimized; is called a *constrained Steiner tree* (or *delay constrained multicast tree*) [35].

A *cost-based shortest path* from node v to destination d is a path from v to d that has the minimum cost. A *delay-based shortest path* from node v to destination d is a path from v to d that has the minimum delay. A *delay-constrained shortest path* is one with the minimum cost among all paths with delays under the delay constraint. A *delay* (*cost*)*-based shortest path tree* rooted at node s to the set S of destination nodes is a tree consisting of delay (cost) based shortest paths from s to each destination $d \in S$. A *feasible* cost (delay)-based shortest path from node v to destination d is a cost (delay) based shortest path from v to d that satisfies the

delay constraint. A *potentially feasible* cost-based shortest path from node v to destination d is a cost based shortest path from v to d where the immediate successor w of v on the path can reach d under the delay constraint; that is, $P(v)+D(v,w)+SD(w,d) < \Delta, SD(w,d) =$ min delay from w to d; $D(v,w) =$ delay on the link (v, w); $P(v) =$ delay on path from s to v in the tree.

2.2. Overview of the Proposed Algorithm

The proposed algorithm takes the following approach to overcome the shortcomings that exist in DKPP and DSPH while maintaining the quality of the multicast tree (i.e., tree cost) as optimal as possible:

1. DKPP only assumes that the link cost to each neighbor is known to the local node and expands the multicast tree edge by edge, which causes a high number of messages and time complexity. Based on the fact that the existing routing information is available at each node, we will use this information to derive the information about the cost-based shortest path from the local node to each node in the network and make the derived information available to the local node so that the SP algorithm can be applied. As in DKPP, it is also assumed that the information about the delay-based shortest path from the local node to each node in the network is known to the local node.

   DSPH is an SP-based algorithm. However, the multicast tree is expanded sequentially to cover each destination. This means that it could take a long time for the algorithm to complete the construction of a multicast tree. The proposed algorithm will expand the multicast tree to cover all destinations in a concurrent manner. The criterion used to expand the multicast tree will be "expand the tree from the local node along the cost-based shortest path if the delay constraint is satisfied; otherwise, expand the tree along the delay-based shortest path." This is different from DSPH's criterion, which is "expand the tree along the cost-based shortest path from the tree to a destination node," that can only be applied sequentially. Also, the criterion that we will use allows the proposed algorithm to include a cost-based shortest path to a destination as much as possible to make the tree cost as low as possible. It is important to note that conflicts may occur if a node is invited more than once to join the tree. The solution is that the node where the conflicts occur will choose its parent whose accumulated delay from the source node is minimal to ensure that all destinations that go through this node will satisfy the delay constraint.
2. DSPH requires that the delay constraint is larger than the maximum delay of all cost-based shortest paths from the source to each destination. Thus, DSPH will not be able to find a delay constrained multicast tree when the delay constraint is smaller than the maximum delay of the cost-based shortest paths from the source to each destination, but larger than the maximum delay of the delay-based shortest paths from the source to each destination (i.e., in the cases that there should exist delay constrained multicast trees). The proposed algorithm will not have such a limitation due to the fact that the tree expansion criterion described in 2 above is used and thus constructs a delay constrained multicast tree even in those cases where DSPH fails to construct one.

The proposed algorithm is called *Distributed Concurrent Shortest Path heuristic* (DCSP) because it tries to cover all destinations concurrently. The concurrency in this context means how the destinations are to be covered, in contrast to the sequential approach used in the existing distributed delay constrained multicast routing algorithms where destinations are covered sequentially one by one.

Informally, given a network $G = (V, E)$, a source node s, a set of destination nodes S, and a delay constraint Δ, DCSP progresses in one or two phases as follows:

2.2.1. Phase I—Setup

1. Starts with a tree containing only the source node *s:*
2. Selects one of its neighbors as a "best" neighbor for each destination node to be covered, and expands the tree concurrently along these best neighbors. The "best" neighbor for a destination node will be the neighbor along which there is a potentially feasible cost-based shortest path that can reach the destination. A *potentially feasible* cost-based shortest path from node v to destination d is a cost-based shortest path from v to d where the immediate successor w of v on the path can reach d under the delay constraint; that is, $P(v) + D(v,w) + SD(w,d) < \Delta$, where $SD(w,d) =$ minimum delay from w to d; $D(v,w) =$ delay on the link (v,w); $P(v) =$ delay on path from s to v in the tree. In addition, $SC(w,d) =$ minimum cost from w to d. Both $SD(w,d)$ and $SC(w,d)$ are available from a lower level protocol such as a unicast routing protocol running on the node [14]. So, the "best" neighbor w of node v is the one satisfying the following constraints: min $SC(w,d)$ and $P(v) + D(v,w) + SD(w,d) < \Delta$;
3. If there is no potentially feasible cost-based shortest path found, adds the destination node under consideration to the list N of destinations that are not yet covered by the tree;
4. If a node is included into the tree by more than one parent node, the node will choose the parent node that has the smallest accumulated delay from the source node;
5. Repeats steps 2–4 until each node in S is either included in the tree or included in the list N of destinations which are not yet covered by the tree.

2.2.2. Phase II—Adjustment

1. Starts from the source node with a tree containing all nodes *S–N* that are included in Phase 1;
2. Selects a "best" neighbor for each destination node to be covered, and expands the tree concurrently along these best neighbors. The "best" neighbor for a destination node in this stage will be the neighbor along which there is a feasible delay based shortest path;
3. If there is no feasible delay-based shortest path found, adds the destination node under consideration to the list M of destinations that are not yet covered by the tree;
4. If a node is included into the tree by more than one parent node, it will choose the parent node that has the smallest accumulated delay from the source node;

5. Repeats steps 2–4 until each node in N is included in the tree or included in the list M of destinations that are not yet covered by the tree.

Phase II will be invoked only if Phase I cannot cover all destination nodes in S. Phase II terminates with two possible outcomes depending on whether the list M of uncovered destinations is empty or not. If M is empty, the algorithm returns a delay constrained multicast tree. Otherwise, the algorithm returns a failure. One could then use the given delay information to build a delay based shortest path tree that can be considered as a nonoptimal delay constrained multicast tree. It can be seen that two phases are required in the proposed algorithm as no backtracking mechanism is used in case there is no potentially feasible cost-based shortest path found during the setup of a delay constrained multicast tree.

2.3. Details of the Proposed Algorithm

The following eight types of messages are used in DCSP:

open—for starting a multicast connection;
setup—for forming a delay constrained shortest path from the source node to a destination node;
break—for notifying a node's parent to break a loop;
reject—for rejecting the invitation to join the tree because of either the violation of constraints or detection of a loop;
destination—for adding the destination to the uncovered destination list;
deny—for denying the addition of the destinations to the tree;
notify—for notifying the source node that a destination node has been covered;
adjust—for adjusting the tree along the delay-based shortest path from the source to a destination node.

The pseudocode of DCSP is shown in Appendix 1. Following are the details of the algorithm:

2.4. Setup Phase

1. When a node receives a request (*open* message) for starting a multicast connection with parameters S and Δ, it becomes the source node s of the multicast connection; hence, the root node in the multicast tree.
   a. The source node then selects a next node among its neighbors for each destination in the set of destinations under consideration by the source node. Denote the set of destinations under consideration by a node v as $S'(v)$. For each destination $d_i \in S'(v)$, a table called *destTable* records the following information: *state*—the local state for d_i and *via*—the "best" neighbor node along which there is a potentially feasible cost-based shortest path to reach d_i. Initially, both fields are "*unknown*." For the source node, all destinations should be included in the set of destinations under consideration. So, $S'(s) = S$. Field *via* for each destination in destTable is set to the "best" neighbor node.
   b. For each selected neighbor (i.e., it appears in field *via* of destTable at least once), a *setup* message is sent to the neighbor in an attempt to include the neighbor into the multicast tree. This *setup* message carries a set of destinations to be covered via the link to the neighbor and the accumulated delay from the source node s. The *state* fields for the corresponding destinations in destTable are set to "*setup*."
2. When a node v receives a *setup* message, it includes itself into the tree.
   a. If the inclusion of node v into the tree introduces a loop (i.e., node v is already in the tree), node v breaks the loop as follows:
      - If the accumulated delay along the existing path from the source node s to all destination nodes to be covered via the link to node v is within the delay constraint, then node v sends a *reject* message to the sender of the *setup* message.
      - Else if the new accumulated delay from the source node s to node v via the sender of the *setup* message is less than the old accumulated delay, then node v sends a *break* message to its parent to break the existing path and accepts the new parent.
      - Otherwise, node v sends a *deny* message to the sender of the *setup* message, so that the sender can reexpand the tree to cover those denied destinations.
   b. If node v is a destination itself, it sends a *notify* message to the source node s.
   c. Regardless, whether node v itself is a destination or not, if there are destinations yet to be considered [i.e., $S'(v) - \{v\}$ isn't empty], then node v sets field *via* for each destination in destTable to the "best" neighbor node, and sends a *setup* message to the selected neighbors to include the neighbors into the multicast tree. This is achieved in a way that is the same as a and b. Note that if there is no potentially feasible cost-based shortest path that can be found for a destination, node v sends a *destination* message to the source node to indicate that the destination cannot be covered in Phase I.
3. When the source node s receives the *destination* message, it adds the specified destination node to the uncovered destination list *US*.
4. When the source node s receives the *notify* message, it adds the received destination node to the covered destination list *CS*.

   If all destination nodes have been processed, the algorithm checks if the uncovered destination list *US* is empty. If *US* is empty, the algorithm terminates. Otherwise, the adjustment phase (Phase II) is invoked.
5. When node v receives the *break* message, it removes the sender node from the multicast tree. It marks the *state* of those destinations in $S'(v)$ that are already in the tree via the sender node as "*broken*." If the sender node is its only child, then the *break* message will be forwarded further to its parent node.
6. When node v receives the *reject* message, it removes the sender node from the multicast tree and marks the *state* of those destinations in $S'(v)$ that have been set up via the sender node as "*rejected*." If the sender node is its only

child, then the *reject* message will be forwarded further to its parent node.

7. When node v receives the *deny* message, it removes the sender node from the multicast tree, resets the *state* of those destinations in $S'(v)$ that are already included in the tree via the sender node as "*unknown*," and marks the link (v, *sender*) as "*unusable*."

   It then expands the tree as in 2.*c*.

2.5. Adjustment Phase

The algorithm in Phase II is very similar to what is described in Phase I, except the following differences:

1. The set of destinations under consideration in the source node is *US* instead of *S*.
2. The criterion for choosing the "best" neighbor node for a destination becomes the neighbor node along which there is a feasible delay based shortest path.
3. The *setup* message is replaced with the *adjust* message.
4. On receiving a *break* or *reject* message, if all destination nodes have been processed and the uncovered destination list is not empty, algorithm terminates with failure.

The following theorem establishes that DCSP generates a delay constrained multicast tree.

Theorem 1. *When DCSP successfully constructs a multicast tree, the constructed tree is delay constrained.*

Proof. In both phases of DCSP, a node v can expand the multicast tree to node w only if, for each $d \in S'(v)$ that is going to be routed via node w, $P(v) + D(v,w) + SD(w,d) < \Delta$. Thus, every node that is included into the multicast tree must satisfy the delay constraint. If node w is already in the tree, it is guaranteed that all destinations that have been chosen to be routed via node w will continue to satisfy the delay constraint because DCSP allows node w to become a child of node v only if the new accumulated delay value $P(w)$ is smaller than the older one. Therefore, if the algorithm successfully constructs a multicast tree, the constructed tree will be delay constrained. ∎

The performance of DCSP can be analyzed in terms of the number of message exchanges, the convergence time, and the cost of the generated multicast trees. The following theorem shows that in the worst case, DCSP has a lower message and time complexity than DKPP, and a lower time complexity than DSPH. The simulation studies reported in the next section will complement the analysis made here by comparing DCSP with DKPP and DSPH further in terms of the number of message exchanges and the convergence time in the average case. The simulation studies also make the comparison among these three algorithms in terms of the cost of the generated multicast trees and the success rate in tree construction.

Theorem 2. *DCSP's message complexity is O(mn) and time complexity is O(n) in the worst case.*

Proof. During the setup phase, the *setup* message for each destination will be sent at most n times. Thus, there will be at most O(mn) *setup* messages because there are m destinations. The *notify*, *destination*, and *deny* messages will not be sent more than m times because only one *notify* or *destination* or *deny* message can be sent for each destination. The *break* or *reject* messages can be sent at most n times along the way to each destination. On the other hand, during the adjustment phase, all types of messages can be sent no more than O(mn) times. So, the algorithm runs with a message complexity of O(mn) messages.

As the algorithm expands the tree to reach all destinations concurrently, the setup (or adjust) message will reach each reachable destination in O(n) time during the setup (or adjustment) phase. So, the algorithm runs with a time complexity of O(n). ∎

Compared with DKPP's message complexity of O(n^3) and time complexity of O(n^3) [19], DCSP has a much better message and time complexity. DSPH runs with O(mn) messages and time as it constructs a tree to cover each destination sequentially. So, DCSP has a better time complexity than DSPH.

It should be noted that the dynamic membership (i.e., set of destination nodes S is changing during a multicast session) can be handled in the proposed algorithm using an approach similar to the one used in the existing distributed delay constrained multicast routing algorithms such as DSPH [14]. That is, when a destination node is added as a new member, the source s can expand the tree along a delay-constrained cost-based shortest path from s to the destination node. A destination node can be removed if the node is a leaf node of the tree, or it can be marked as a nondestination node if the node is a nonleaf node of the tree.

3. AUGMENTING DCSP WITH FAULT RECOVERY

3.1. Overview of the Proposed Augmentation

Like the existing distributed multicast routing algorithms, DCSP assumes that there are no network topology changes during the construction of a multicast tree and the on-going multicast session. If the network topology changes, DCSP will fail to complete the delay constrained multicast tree. DCSP depends on a naive fault recovery approach, which simply waits for applications to restart the algorithm from scratch. This makes DCSP, like DKPP and DSPH, take longer time to complete when failures occur.

Our aim is to design an effective and efficient fault recovery approach, which will perform the fault recovery transparent to the applications, will not interrupt the running traffic on the unaffected portion of the multicast tree, and

will shorten the time to recover from node failures that may occur during the construction of a multicast tree or during an on-going multicast session. We also want the fault recovery approach to be integrated into DCSP such that the augmented DCSP recovers from node failures by adaptively constructing a delay constrained multicast tree whenever node failures occur.

In addition, we want DCSP augmented with the fault recovery approach, called henceforth *Adaptive distributed Concurrent Shortest Path heuristic* (ACSP), to generate delay constrained multicast trees whose quality is as good as that of DCSP in terms of tree cost, and perform as well as DCSP in terms of message complexity. Note that, in general, these two goals conflict with each other. That is, it is generally anticipated that extra messages will be needed to be able to perform fault recovery.

The basic idea underlying our fault recovery approach is to avoid rebuilding the entire multicast tree again by localizing the recovery actions to the failed portion of the multicast tree. To implement this idea there should be a mechanism to communicate the information about the failed subtree to the nodes that are participating in the computation so that this information can be taken into account by these nodes during the rest of the tree construction process. A straightforward approach is to flood the network with this information. However, flooding requires $O(n^2)$ messages just for the notification of the node failure alone. In ACSP, the failed subtree is not allowed to flood the network with fault information that is propagated instead through the regular tree setup and follow-up messages as needed.

ACSP progresses in the same way as DCSP with respect to the main steps of the algorithm except that ACSP checks between the major steps if any tree node has failed. When a node failure is detected, ACSP removes the subtree rooted at the failed node and notifies the source node s with the id's of the destination nodes that were covered by the removed subtree. Hence, the source node s is enabled to add these removed destination nodes when all the remaining destination nodes have been added to the multicast tree. It is important to note that loops may be introduced into the multicast tree due to the changes in the network topology. ACSP detects the existence of a loop by checking if a node (say v) to be added is already in the tree. The loop removal is achieved by choosing a path from the source node s to node v, which has the minimum delay. This strategy ensures that all existing paths from the source node s to destinations that go through node v still satisfy the delay constraint. If the new parent of v has the minimum delay from the source node s, node v accepts the new parent and breaks itself from its previous parent. Otherwise, node v rejects the new parent. Hence, the loop is removed in either case. In addition, ACSP continues to check node failures and takes recovery steps to repair the multicast tree if node failures occur during an on-going multicast session.

Informally, given a network $G = (V, E)$, a source node s, a set of destination nodes S, and a delay constraint Δ, ACSP progresses in one or two phases as follows:

3.1.1. Phase I—Setup. The same as in Phase I of DCSP except the following additional step (new step 5) that is inserted between steps 4 and 5 of Phase I of DCSP:

5. Checks if a tree node fails. When a failure is detected, removes the subtree rooted at the failed node and returns the destination nodes covered by the subtree back to the list of destinations to be covered by the tree;

3.1.2. Phase II—Adjustment. The same as in Phase II of DCSP except the following additional step (new step 5) that is inserted between steps 4 and 5 of Phase II of DCSP:

5. Checks if a tree node fails. When a failure is detected, removes the subtree rooted at the failed node and returns the destination nodes covered by the subtree back to the list of destinations to be covered by the tree;

3.2. Details of the Augmentation

Besides the eight types of messages in DCSP, a new type of message is used in ACSP, which is *remove*—for removing a subtree from the tree.

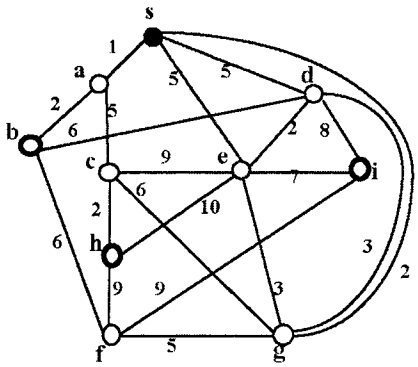
It is assumed that node failures are detected by a lower level protocol. Following are the details of ACSP where the augmentation of DCSP is given in bold:

3.3. Setup Phase

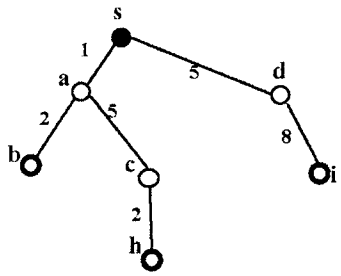
1. and 2. The same as Steps 1 and 2 of the algorithm for Phase I of DCSP.
3. When the source node s receives the *destination* message, **it adds the specified destination node to the uncovered destination list *US* if no feasible path can be found or to the failed destination list *FS* if a node failure occurred.**

   If all destination nodes have been processed, **the algorithm checks if *US* and *FS* are empty**. If this is the case, the algorithm terminates. **If *FS* is nonempty, it expands the tree as in 1**. Otherwise, the adjustment phase (Phase II) is invoked.
4. When the source node s receives the *notify* message, it adds the received destination node to the covered destination list *CS*.

   If all destination nodes have been processed, **the algorithm checks if *US* and *FS* are empty.** If this is the case, the algorithm terminates. **If *FS* is nonempty, it expands the tree as in 1.** Otherwise, the adjustment phase (Phase II) is invoked.
5. 6. and 7. The same as Steps 5, 6, and 7 of the algorithm for Phase I of DCSP.
8. **When node v in the tree detects that its child node y fails, v removes y from the tree**.
9. **When node v in the tree detects that its parent node fails or receives the *remove* message, node v removes itself from the tree. If node v is not a leaf node, it will forward the *remove* message to all of its children. If node v is a destination node, it sends a *destination* message to the source node s so that the source node s will add the destination node back to *FS*.**



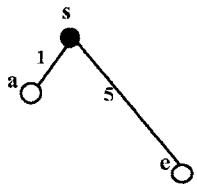
(a) a network.



(b) a multicast tree for (a).

dest	*cost*	*via*	*state*
b	3	a	unknown
h	8	a	unknown
i	12	e	unknown

(c) initial destTable table.



(d) partial multicast tree for (a).

dest	*cost*	*via*	*state*
b	2	b	setup
h	7	c	setup

(e) destTable table updated by node *a*.

dest	*cost*	*via*	*state*
i	7	i	setup

(f) destTable table updated by node *e*.

FIG. 1. An example for ACSP Algorithm.

3.4. Adjustment Phase

The algorithm in Phase II is very similar to what is described in Phase I, except the following differences:

1. The set of destinations under consideration in the source node is *US* instead of *S*.
2. The criterion for choosing the "best" neighbor node for a destination becomes the neighbor node along which there is a feasible delay based shortest path.
3. The *setup* message is replaced with the *adjust* message.
4. On receiving a *break* or *reject* message, if all destination nodes have been processed and the uncovered destination list is not empty, the algorithm terminates with failure.

An example network shown in Figure 1 is used to illustrate the algorithm. For clarity of the diagrams, the same integer number is used to represent both cost and delay values. Node *s* is the source node and the dark nodes *b*, *h*, and *i* are destination nodes. Δ is 14 and node *e* fails after node *e* is covered.

1. Step 1: when node *s* receives an *open* message, it becomes the first node in the tree (see Fig. 1b).
2. Step 1.1: node *s* calculates the initial destTable as shown in Figure 1c.
3. Step 1.2: node *s* sends a *setup* message to both node *a* and *e* based on Figure 1c.
4. Step 2: when node *a* and *e* receive the *setup* message, both node *a* and *e* will be included in the multicast tree (see Fig. 1d). Nodes *a* and *e* will update destTable as in Figure 1e and f.
5. Suppose that node *e* fails. So. node *e* will be removed from the multicast tree. while node *a* will send a *setup* message to node *b* and *c*.
6. Upon receiving the *setup* message from node *a*, node *b* and *c* will be included in the multicast tree (Step 2). Also, node *b* is going to send a *notify* message to the source node *s* and node *c* will send a *setup* message to node *h*.
7. Upon receiving the *setup* message from node *c*, node *h* will be included in the multicast tree (Step 2). Also, node *h* is going to send a *notify* message to the source node *s*.
8. The source node *s* will detect that destination node *i* is still not covered. So, the algorithm will enter Phase II and node *i* will be covered by the path *s*, *d*, and *i*.
9. The final multicast tree for the example is shown in Figure 1b.

The correctness of ACSP is established by the following theorems.

Theorem 3. *The delay-constrained multicast tree built by ACSP is free from loops.*

Proof. For a loop to exist in the multicast tree, there must be a node that has two parent nodes. In ACSP, each time a node receives a *setup* message for joining the tree, the node checks if it is already in thc tree. If it is not in the tree yet, it

will join the tree and set the sender of the *setup* message as its parent node. Otherwise, it will either reject the *setup* message or accept the *setup* message but break from its existing parent node first. Thus, in either case, each node has one and only one parent node and therefore the delay-constrained multicast tree built by ACSP is free from loops. ∎

Theorem 4. *When ACSP successfully constructs a multicast tree, the constructed tree is delay constrained.*

Proof. The proof is similar to the proof of Theorem 1.∎

As in the case of DCSP, the performance of ACSP can be analyzed in terms of the number of message exchanges, the convergence time, and the cost of the generated multicast trees. It was established by Theorem 1 that, without node failures, DCSP runs with O(mn) message and O(n) time complexity in the worst case. That is, in the worst case, DCSP has a lower message and time complexity than DKPP, and a lower time complexity than DSPH. However, because each node failure will make DCSP rerun, DCSP will have O(kmn) message and O(kn) time complexity if there are $k-1$ node failures during a multicast session. The following theorem establishes that in the worst case, ACSP performs as good as DSPH in terms of the message and time complexity. The simulation studies reported in the next section will complement the analysis here by comparing ACSP with DCSP further in terms of the number of message exchanges and the convergence time in the average case. The simulation studies also make the comparison among these two algosrithms in terms of the cost of the generated multicast trees and the success rate in tree construction.

Theorem 5. *Suppose that $k-1$ is the total number of node failures occurring in the network during the construction of a delay constrained multicast tree and during the on-going multicast session thereafter. Then ACSP's message complexity is O(kmn) and time complexity is O(kn) in the worst case.*

Proof. In the presence of $k-1$ node failures, ACSP is forced to try k times to complete the construction of a delay constrained multicast tree. Each time, in the worst case, O(n) *setup* messages will be sent for each destination because there are n nodes in the network. Thus, in the worst case, a total of O(mn) *setup* messages will be sent because there are m destinations in the network.

Because only one *destination* message can be sent for each destination, the total number of *destination* messages sent will not be more than m.

In the worst case, O(n) *break* or *reject* messages can be sent along the way to each destination.

Because *remove* messages can only be sent once along each edge of the subtree rooted at the failed node, a total of O(n) *remove* messages can be sent, in the worst case.

Therefore, the algorithm's message complexity is O(kmn).

The setup message will reach all destinations in O(n) time for each of the k trials of completing a delay-constrained multicast tree. Therefore, the algorithm's time complexity is O(kn). ∎

4. PERFORMANCE ANALYSIS

Two series of simulations have been performed to compare the performance of DCSP with DKPP and DSPH, and the performance of ACSP with DCSP. These simulations were carried out by applying DCSP, DKPP, DSPH, and ACSP to networks generated by Waxman's approach [37] and by using the setup proposed in [35]: the nodes in a network are randomly distributed over a rectangular coordinate grid. Each node is placed at a location with integer coordinates. The Euclidean metric is then used to determine the distance between each pair of nodes. A link between two nodes u and v is added with a probability that is given by the function $P(u,v) = \beta \exp(-d(u,v)/\alpha L)$, where $d(u,v)$ is the distance from u to v, L is the maximum distance between any two nodes, and $0 < \alpha \leq 1, 0 < \beta \leq 1$. Larger values of β result in graphs with higher link densities, while small values of α increase the density of short links relative to longer ones. The α and β values used in our simulation study are 0.7 and 0.7, respectively. The cost of a link (u,v) in the graph is the distance between nodes u and v on the rectangular coordinate grid. The delay of a link is assigned to a value that is uniformly distributed over the range between 0 and 60. Graphs are generated and tested until the graph is a two-connected network, which has at least two paths between any pair of nodes. The random graphs do have some of the characteristics of an actual network. It has been shown by simulation that the performance of a multicast routing algorithm when applied to a real network is almost identical to its performance when applied to a random two-connected network [22]. The number of messages exchanged, the convergence time and the cost of the generated multicast trees are measured by their average value in a total of 100 simulation runs on a network with 200 nodes. Δ is set to $d_{\max} + (i/8)d_{\max}$ where $d_{\max} = \max(\{d_u \mid \text{for any } u \in S : d_u \text{ is the delay on the delay based shortest path from } s \text{ to } u\})$ and i is an integer between 1 and 15.

In this study, an exchanged message is only counted once from its sender to its receiver regardless of the number of intermediate nodes as long as the algorithm running on such a node does not interpret the message. Also, the convergence time is calculated by taking one message exchange as a time unit, not the CPU time, for any of the algorithms that were implemented in C and run on a Solaris 5, Sun SPARC workstation. Note that, although within one time unit, there may be several message exchanges occurring in the network, multiple message exchanges within the same time unit are considered as one message exchange when calculating the convergence time.

4.1. DCSP versus DKPP and DSPH

The first group of simulations are conducted by fixing the delay at $i = 3$ and letting the size of a multicast group change

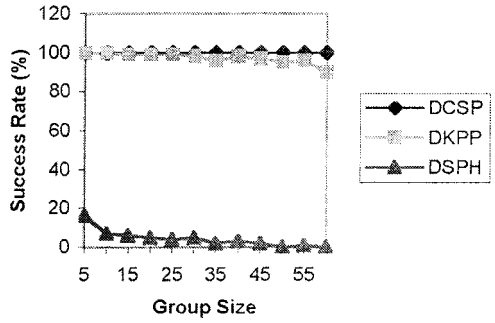


FIG. 2. Success rate versus group size when delay is fixed at $i = 3$.

from 5 to 60 in order to see how the algorithms perform when the group size changes. Figures 2–5 show the success rate, tree cost, number of messages, and time versus group size. From Figure 2, we can see that under the delay constraint of $i = 3$. DSPH has a near-zero percent success rate for almost all group sizes. This is a serious problem for DSPH, which means that DSPH is not useful when the delay constraints are tight. As a result, DSPH cannot be compared in the following three figures for tree cost, number of messages and time, and therefore. DSPH is not plotted in these three figures. Figure 3 shows that when compared with the tree cost of SPTd, which could be considered as the delay constrained multicast tree without any optimization on tree costs, DKPP and DCSP give a reduction of 20%. This shows that the optimization of the cost of multicast trees is worthwhile.

From Figure 3, we also see that the tree costs generated by DCSP and DKPP are almost identical. However, there is a big difference between DCSP and DKPP in terms of the number of messages and time, as can be seen in Figure 4 and Figure 5, respectively. The number of messages used by DKPP is between 705 and 987 times more than that by DCSP, while the time required by DKPP is between 82 to 110 times more than that by DCSP. Figure 2 indicates that the larger the group size, the worse the success rate of DKPP, while DCSP almost has 100% success rate for all group sizes. So, it can be concluded that the performance of DCSP is much better than that of DKPP in terms of success rate, number of messages, and time.

The second group of simulations are conducted by fixing the group size at 20 nodes and letting the delay change from $i = 1$ to 15 to see how the algorithms perform when the

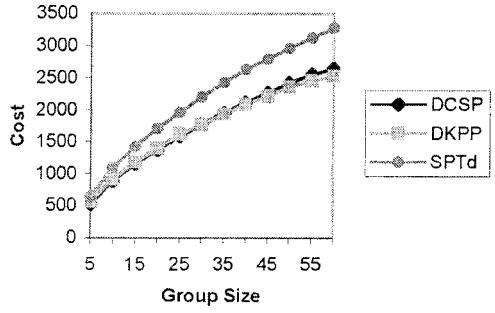


FIG. 3. Tree cost versus group size when delay is fixed at $i = 3$.

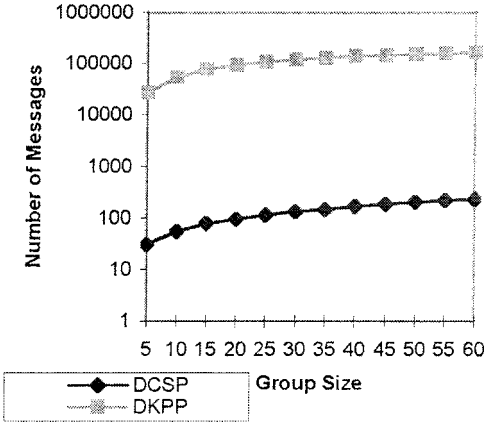


FIG. 4. Number of messages versus group size when delay is fixed at $i = 3$.

delay constraint changes. Figures 6–9 show the success rate, tree cost, number of messages, and time versus the delay constraints, respectively. Figure 7 shows that the costs of multicast trees generated by DCSP are as close to optimal as DKPP. However, the number of messages used by DKPP is between 409 and 1734 times more than that by DCSP (see Fig. 8), while the time needed by DKPP is between 56 and 364 times more than that by DCSP (see Fig. 9). Also, with respect to the success rate observed in our simulation study as shown in Figure 6, DKPP may fail to construct a multicast tree under tight delay constraints, while DCSP can always successfully construct a delay constrained multicast tree. As for the tree cost, Figure 7 shows that the cost of multicast trees generated by DSPH is better than both DKPP and DCSP. But, the success rate of DSPH is very low under the tight delay constraint, which undermines the optimality of its tree costs. Further, the time complexity of DSPH is much higher than that of DCSP, while DCSP has similar number of messages exchanged to that of DSPH. In summary, compared with DKPP, DCSP had better performance in terms of number of messages, time, and success rate, while producing equal quality of multicast trees in terms of tree cost. Compared with DSPH, DCSP has the advantage in time and success rate, while performing as well

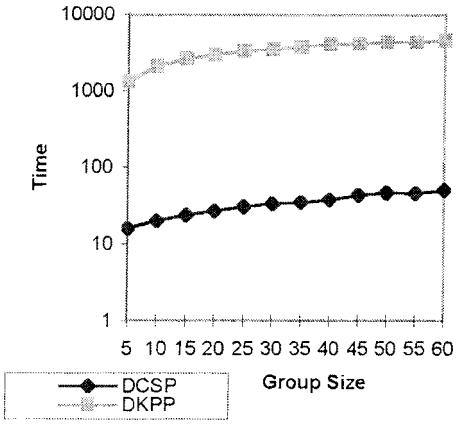


FIG. 5. Time versus group size when delay is fixed at $i = 3$.

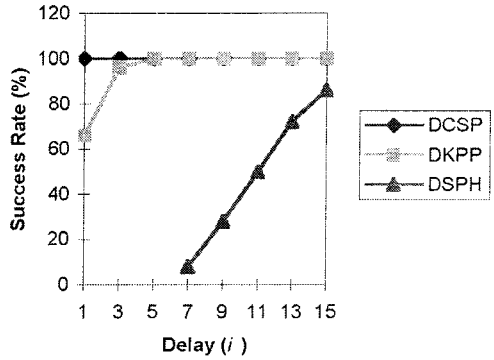


FIG. 6. Success rate versus delay when group size = 20.

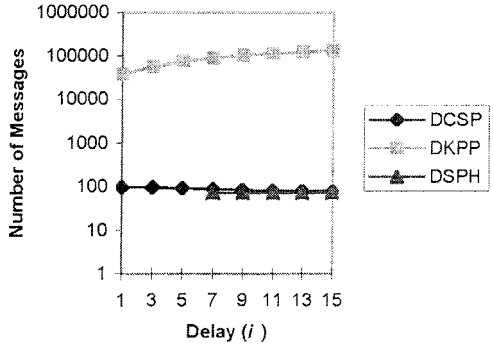


FIG. 8. Number of messages versus delay when group size = 20.

in terms of number of messages exchanged. The better tree cost of DSPH is obtained at the cost of a very poor success rate.

4.2. ACSP versus DCSP

The performance of ACSP is compared with that of DCSP in the average case under the condition that a node failure occurs during the construction of a delay constrained multicast tree or during an on-going multicast session. This comparison is carried out in a setup proposed by [35]. Accordingly, DCSP uses the naïve fault recovery approach; that is, DCSP has to be rerun from scratch when a node failure occurs. Two types of simulations were conducted by injecting node failures into the networks: one type is for the case when node failures occur during the construction of a multicast tree, and the other type is for the case when node failures occur during an on-going multicast session. For the first type of simulation, at most one node failure is allowed to occur during the construction of the multicast tree. The timing for a node failure is randomly selected so it could occur randomly among the different stages of the construction of the multicast tree. The failed node is randomly selected among the nodes in the multicast tree built so far that are neither the source node nor the destination nodes when node failure occurs. This is motivated by the fact that the failure of the source node means that there will be no multicast tree to be built and the failure of a destination node means that the constructed multicast tree will not be comparable with other multicast trees that cover all the destination nodes. For the second type of simulation, DCSP has to be rerun to rebuild the entire multicast tree when a node failure occurs during an on-going multicast session. Again, the failed node is randomly selected among the nodes in the multicast tree during a multicast session. Because DCSP will be rerun when a node failure occurs while ACSP will always return a multicast tree no matter whether a node failure occurs or not, it is expected that ACSP will have a better success rate than DCSP. The results of the performance analysis given in this section should be interpreted with the understanding that a single node failure is considered when running the simulations.

Figures 10–12 show the simulation results when Δ is set to $d_{\max} + (3/8)d_{\max}$ where $d_{\max} = \max(\{d_u \mid \text{for any } u \in S : d_u$ is the delay on the shortest path from s to $u\})$, and the group size changes between 5 and 60 in 200-node networks. In the figures, suffix "$-c$" means during the construction of a multicast tree, suffix "$-m$" means during the on-going multicast session and "SPT-d" means the delay based SPT. The delay based SPT could be considered as the delay constrained multicast trees without any optimization on tree costs. Figure 10 shows that the costs of the trees generated by ACSP are almost identical to those by DCSP. This result is very encouraging. DCSP calculates the multicast tree based on the consistent current network topology information after it reruns while ACSP might use different network topology information for different parts of a delay constrained multicast tree. Intuitively, it could be expected that the costs of the trees generated by ACSP should be noticeably higher than those by DCSP. But, the simulation results show that it is not

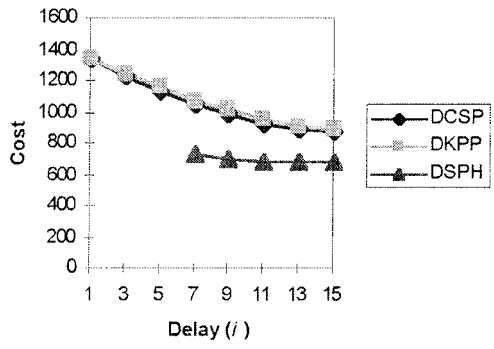


FIG. 7. Tree cost versus delay when the group size = 20.

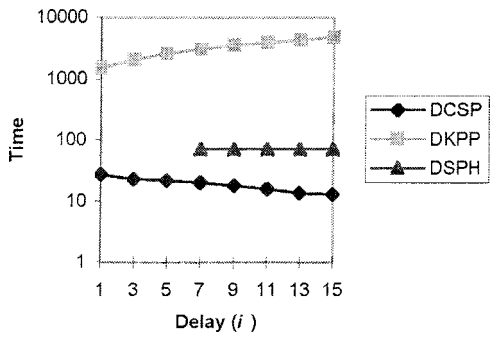


FIG. 9. Time versus delay when group size = 20.

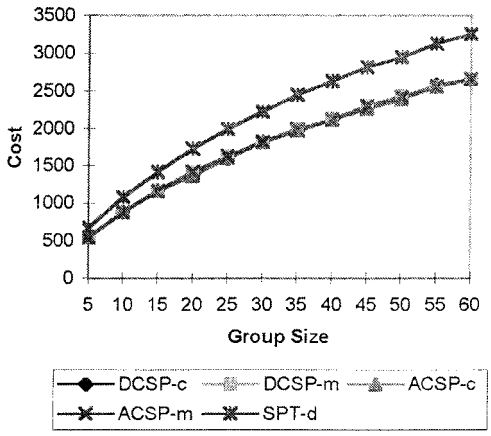


FIG. 10. Tree cost versus group size when node failure occurs.

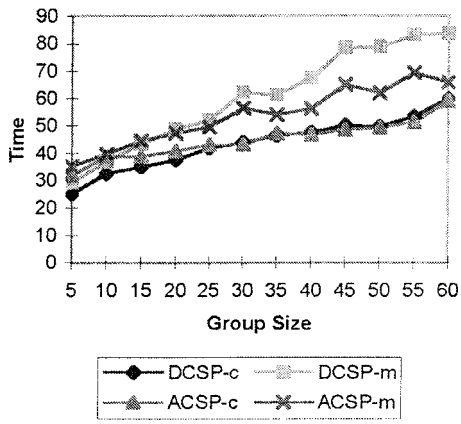


FIG. 12. Time versus group size when node failure occurs.

the case. It is also shown in Figure 10 that the delay constrained multicast tree algorithms generate trees with much better cost performance than the algorithms without considering optimization on the tree cost such as SPT-d (note that in Fig. 10 the top curve is for SPT-d, whereas the bottom curve is for the other algorithms). This means that it is worth using the delay constrained multicast tree algorithms rather than using SPT-d directly. Figure 11 shows that the number of messages required by DCSP is up to 28% more than that required by ACSP during the construction of a delay constrained multicast tree, and is up to 70% more than that required by ACSP during the on-going multicast session. This result is surprising as we know that doing fault recovery normally costs extra number of messages. Intuitively, one would expect that because DCSP is very efficient in using messages, any fault recovery approach that tries to merge the list of uncovered destinations in the failed subtree with the list of uncovered destinations in the participating node will have a higher number of messages than DCSP even though DCSP has to run twice. Contrary to this expectation, ACSP has a significantly better performance than DCSP in terms of the number of messages. This is due to the fact that ACSP uses the approach that refrains from sending messages on node failure. The previous analysis shows that the delay on sending node failure messages does not have a negative effect on the quality of the generated multicast trees. Figure 12 shows that the convergence time required by DCSP is up to 27% more than that by ACSP during the on-going multicast session. It confirms what has been expected. Through the localized recovery approach taken by ACSP, it is expected that the convergence time can be reduced by ACSP.

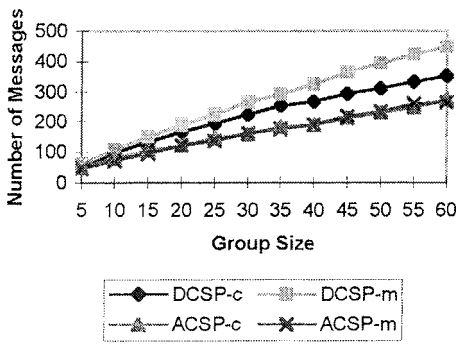


FIG. 11. Number of messages versus group size when node failure occurs.

5. RELATED WORK

In addition to the centralized and distributed QoS-based multicast routing algorithms mentioned in the Introduction, there are also many QoS-based protocols that have been proposed in the literature, such as YAM (or spanning join) [6], QoSMIC [11], and QMRP [8]. They are all based on the multiple-paths approach [26]; that is, the protocols will generate multiple paths between the existing multicast tree and a destination node and then the protocol will pick the best one based on certain QoS criteria to connect the destination node into the tree. The multiple-paths approach is mainly used to trade message overheads and system resources for high success rate. Other work includes the delay constrained group multicast routing problem [21], which finds a set of constrained Steiner trees for each member in a multicast group, the multiple QoS constraints multicast routing problem [10, 27, 36] which finds the multicast trees satisfying multiple QoS constraints and load balancing in various networks for load distribution [12, 30]. An excellent survey on many QoS-based algorithms is given in [32].

Algorithms have also been proposed to address the dynamic multicast routing problem [7, 32, 37] where the multicast group can be dynamic, that is, new destination nodes (i.e., group members) may be added to the group, and existing destination nodes may be removed from the multicast group. The algorithm in [31] is a centralized dynamic multicast routing algorithm with the rearrangements of the tree, while the algorithm proposed in [1] is a distributed algorithm that is able to consider the membership changes during the tree building period and the concurrent multiple membership changes. In addition to the dynamic group membership, the dynamic multicast routing environment can also mean the network topology changes [25]. For example, a new node is

added, an existing node or link fails, or the cost values associated with links change. So, some fault recovery approaches are required in such a dynamic multicast routing environment. To distinguish the network topology dynamics from the group membership dynamics, we call the multicast routing problem that takes the topology changes into consideration as the *topology dynamic multicast routing problem*.

Several fault recovery approaches for unconstrained multicast routing problem have been proposed in the literature. The one specified in [2,3] uses the approach in which the disconnected subtree is flushed and all members in the subtree attempt to rejoin the tree individually, which may cause a substantial increase in network traffic as the control messages are propagated through the network. The one described in [29] uses a "reversing tree edges" method to reconnect the disconnected subtree with the multicast tree to reduce the traffic of control messages. It is stated in [29] that reconnecting the subtree using this approach may not always generate loop free multicast trees, so their protocol flushes the subtree in these situations. Both approaches described above are for the receiver-initiated multicast routing algorithms, specifically the core based tree (CBT) protocol as opposed to the sender-initiated multicast routing algorithm studied in this article. The fault recovery approach proposed in this article adopts the approach taken in [35] to the proposed algorithm for constructing a multicast tree in a concurrent fashion. Reference [35] gives an algorithm and its related fault recovery approach, which build a multicast tree sequentially one path at a time. In contrast, the algorithm and its fault recovery approach proposed in this article build all paths in a multicast tree concurrently.

6. CONCLUDING REMARKS

In this article, we have proposed an efficient distributed delay constrained multicast routing algorithm that constructs a multicast tree to cover all destination nodes in a concurrent manner. It has been observed that the existing algorithms have several deficiencies. An MST based algorithm like DKPP runs with a high complexity of $O(n^3)$ in the number of messages and time, and has a very low success rate in constructing a delay constrained multicast tree especially under tight delay constraints. On the other hand, an SP based algorithm like DSPH has a lower complexity of $O(mn)$ in the number of messages and time, but has a fatal deficiency; that is, the algorithm will not be able to find a solution if the delay constraint is less than the maximum delay of a cost based shortest path tree. Second, both algorithms try to construct the multicast tree sequentially without taking advantage of the concurrency in the underlying distributed computation. Last, both algorithms have to be restarted when node failures occur during the multicast tree construction period or during the on-going multicast session.

The proposed algorithm (DCSP) builds the multicast tree in a concurrent manner that results in a very low time complexity of $O(n)$. It also has a low complexify of $O(mn)$ in the number of messages, high success rate, and high quality of multicast trees in terms of tree costs. Compared with DKPP, the proposed algorithm gives better performance in terms of the success rate, the tree cost, the number of exchanged messages, and the convergence time. It is interesting to see that the high complexity of DKPP does not translate into better quality of trees (i.e., smaller tree cost). Compared with DSPH, the proposed algorithm solves the delay constrained multicast tree problem in the cases where DSPH is not able to. Also, it gives better performance in terms of convergence time and performs as well in terms of the number of message exchanges. Both advantages could be critical to some real-time applications, which require the construction time for a multicast tree to be short and the delay constraint to be tight.

The proposed algorithm is augmented with a fault recovery approach (ACSP) that takes into account the changes in the topology of the network. The augmented algorithm can recover from node failures during construction of a delay constrained multicast tree, and during an on-going multicast session without requiring rebuilding of the entire multicast tree. Furthermore, compared with algorithms such as DCSP that use the naïve fault recovery approach, the augmented algorithm (ACSP) gives better performance in terms of the number of exchanged messages and convergence time when applied to a network where node failures occur.

A natural extension of our DCSP is a delay constrained dynamic multicast routing algorithm [called *Distributed Dynamic Concurrent Shortest path heuristic* (DDCS)]. This extension deals with the case where the dynamic group membership changes could occur during the construction of a multicast tree or after a multicast routing tree is constructed. It is assumed that the network could be organized in a hierarchical structure such that an existing delay constrained multicast subtree can join the new group as a whole subgroup without rebuilding the subtree. If a single new destination wants to join the multicast tree, it will send a join message to the source node of the tree. The source node then initiates the process to expand the tree to cover the new destination node in the same way as it covers a set of destination nodes. The criterion that will be used to expand the multicast tree will be "expand the tree from the local node along the cost-based shortest path to a destination node if the delay constraint is satisfied; otherwise expand the tree along the delay-based shortest path to the destination node." This process can be completed in $O(n)$ messages and time.

If a subtree of new destination nodes wants to join the multicast tree, the root node of the subtree will send a join message to the source node of the tree. The join message will carry the maximum delay time in the subtree (say Δ') requesting to join the tree. The source node then initiates the process to expand the tree to cover the new subtree of destination nodes in the same way as it tries to cover a single new destination node, except that the delay constraint becomes $\Delta - \Delta'$ instead of Δ. If an existing destination node wants to leave the multicast tree, it sends a leave message towards the source node along the tree and the nodes on the path will be removed until a fork node or another destination node is reached.

ACKNOWLEDGMENTS

The authors thank the anonymous referees for their useful suggestions, Dr. G. Luo of Nortel Networks for many insightful discussions, and Prof. X. Jia at City University of Hong Kong, Prof. D.S. Reeves at North Carolina State University, and Prof. B.M. Waxman at Southern Illinois University for providing their simulation software to us.

APPENDIX 1: PSEUDO-CODE OF DCSP ALGORITHM

```
/* local = local node                                  */
variables:
/* s = source node                                     */
/* D = accumulated delay from the source               */
/* S' = destinations under consideration               */
/* destTable = node to destination table               */
/*   via - selected neighbor node                      */
/*         for reaching the destination                */
/*   state - local state for the destination           */
/* RouteC = route table by cost                        */
/* RouteD = route table by delay                       */
/*   Both RouteC and RouteD have                       */
/*   next - next hop node                              */
/*   cost - the distance to a destination              */
/*   delay - the delay to a destination                */
/* destS = set of destination nodes                    */
/* US = list of destinations to be covered             */
/* CS = list of destinations covered                   */
initialize:
D := 0:
Set via and state in destTable as unknown;
PHASE I: Setup
/* Transaction 1: */
on receiving open(S, Δ):
   s := local;
   add s to the tree;
   D := 0:
   S' := S;
   /* Transaction 1.1: select neighbors for expansion */
   for each d_i ∈ S' and
      destTabel(d_i).state = unknown do
      destTable(d_i).via := neighbor n_i via which
      there is a potentially feasible cost based
      shortest path reaching d_i:
   end
   /* Transaction 1.2: expand the tree */
   for each n_j : ∃ d_i ∈ S', n_j = destTable(d_i).via do
      destS := {d_i | n_j = destTable(d_i).via, d_i ∈ S'}
      D' = D + delay(local, n_j);
      send setup(destS, D', s. S. Δ) to n_j,
      destTable(d_i).state := setup;
   end

/* Transaction 2: */
on receiving setup(destS, D, s. S, Δ)
   destS := setup.destS;
   /* Transaction 2.1: loop checking */
   if node local is already in the tree then
      /* 2.1.1: reject the setup request */
      if ∀ dest∈destS. D + RouteD(dest).delay < Δ
      then
         send reject() to sender;
         S' := S' ∪ destS;
      /* 2.1.2: accept the new setup request and
         break from the old tree */
      else if setup.D < D then
         send break() to parent;
         add node local to the tree:
         D := setup.D;
         S' := S' ∪ destS;
      /* 2.1.3: the delay constraint may be violated thus
         deny the setup request */
      else
         send deny(destS) to sender;
      endif
else /* It is ok to add node local to the tree */
/* Transaction 2.2: destination reached */
   add node local to the tree;
   D := setup.D;
   If ∃dest ∈ destS, local = dest then
      send notify(dest) to s;
      S' := S'-{dest};
   endif
endif
/* Transaction 2.3: select neighbors for expansion and
      expand the tree. */
   execute Transaction 1.1 and 1.2:
   /* no neighbor found */
   for each d_i ∈ S':destTable(d_i). via = unknown
   do
      destS := {d_i}
      send destination(destS) to s;
      destTable(d_i).state := infeasible;
   end

/* Transaction 3: */
on receiving destination(destS)
   add destS to US
   if CS ∪ US = S then
      if US = Ø then stop;
      else enter phase II;
      endif
   endif

/* Transaction 4: */
on receiving notify(dest)
   add dest to CS
   if CS ∪ US = S then
      if US = Ø then stop;
      else enter phase II;
      endif
   endif
```

```
/* Transaction 5: */
on receiving break()
   remove sender from the tree
   for each d_i: destTable(d_i).via = sender do
      destTable(d_i).state := broken;
   end
if local has no child in the tree then
   send break() to parent;
endif

/* Transaction 6: */
on receiving reject()
   remove sender from the tree
   for each d_i: destTable(d_i).via = sender do
      destTable(d_i).state := rejected;
   end
   if local has no child in the tree then
      send reject() to parent;
   endif

/* Transaction 7: /*
on receiving deny(destS)
   remove sender from the tree
   for each d_i: destTable(d_i).via = sender do
      destTable(d_i).state := unknown;
   end
   set state of (local, sender) as unusable;
   execute Transaction 2.3.
```

REFERENCES

[1] F. Adelstein, G. Richard, and L. Schwiebert, Distributed multicast tree generation with dynamic group membership, Comput Commun 26 (2003), 1105–1128.

[2] A. Ballardie, Core based trees (CBT version 2) multicast routing protocol specification, RFC 2189, Internet Engineering Task Force, September 1997.

[3] A. Ballardie, B. Cain, and Z. Zhang, Core based trees (CBT version 3) multicast routing protocol specification. Internet Draft drafl-ietf-idmr-cbt-spec-v3-01, Internet Engineering Task Force, August 1998.

[4] F. Bauer and A. Varma, Distributed algorithms for multicast path setup in data networks, IEEE/ACM Trans Networking 4 (1996), 181–191.

[5] D. Bertsekas and R. Gallager, Data networks, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1992.

[6] K. Carlberg and J. Crowcroft, Building shared trees using a one-to-many joining mechanism, ACM SIGCOMM Comput Commun Rev 27 (1997), 5–11.

[7] D. Chakraborty, G. Chakraborty, and N. Shiratori, A dynamic multicast routing satisfying multiple QoS constraints, Int J Network Manage 5 (2003), 321–335.

[8] S. Chen, K. Nahrstedt, and Y. Shavitt, A QoS-aware multicast routing protocol, IEEE J Selected Areas Commun 18 (2000), 2580–2592.

[9] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, Introduction to algorithms, MIT, Cambridge, MA, 1992.

[10] Y. Cui, K. Xu, and J. Wu, Precomputation for multi-constrained QoS routing in high-speed networks, Proc IEEE INFOCOM, 2003, pp. 1414–1424.

[11] M. Faloutsos, A. Banerjea, and R. Pankaj, QoSMIC: Quality of service sensitive multicast internet protocol, Proc ACM SIGCOMM 1998, pp. 144–153.

[12] B. Fortz and M. Thorup, Optimizing OSPF/IS-IS weights in a changing world, IEEE J Selected Areas Commun 4 (2002), 756–767.

[13] R.G. Gallager, P.A. Humblet, and P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans Prog Lang Syst 5 (1983), 66–77.

[14] X. Jia, A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks, IEEE/ACM Trans Networking 6 (1998), 828–837.

[15] R.M. Karp, "Reducibility among combinatorial problems," Complexity Computer Communications, R.E. Miller and J.W. Thatcher (Editors), Plenum Press, New York, 1972, pp. 85–103.

[16] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Multicasting for multimedia applications," Proc IEEE INFOCOM, 1992, pp. 2078–2085.

[17] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, Multicast routing for multimedia communication, IEEE/ACM Trans Networking 1 (1993), 286–292.

[18] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Two distributed algorithms for the constrained Steiner tree problem," Proc 2nd International Conference on Computer Communications and Networking, 1993, pp. 343–349.

[19] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, Optimal multicast routing with quality of service constraints, J Network Syst Manage 2 (1996), 107–131.

[20] L. Kou, G. Markowsky, and L. Berman, A fast algorithm for Steiner trees, Acta Informatica 15 (1981), 141–145.

[21] C.P. Low and X. Song, On finding feasible solutions for delay constrained group multicast routing problem, IEEE Trans Comput 51 (2002), 581–588.

[22] C.A. Noronha and F.A. Tobagi, "Evaluation of multicast routing algorithms for multimedia streams," Proc IEEE International Telecommunication Symposium, Rio de Janeiro, Brazil, August 1994, pp. 390–397.

[23] Sanjoy Paul, Multicasting on the internet and its applications, Kluwer Academic Publishers, Norwell, MA, 1998.

[24] R. Prim, Shortest connection networks and some generalizations, Bell Systems Tech J 36 (1957), 1389–1401.

[25] M. Ramalho, Intra- and inter-domain multicast routing protocols: A survey & taxonomy, IEEE Commun Surveys Tutorials, Electronic Mag: http://www.coinsoc.org/pubs/suiveys 3 (2000), pp. 2–25.

[26] N. Rao and S. Batsell, "QoS routing via multiple paths using bandwidth reservation," Proc IEEE INFOCOM, 1998, pp. 11–18.

[27] A. Roy, N. Banerjee, and S. Das, "An efficient multi-objective QoS-routing algorithm for wireless multicasting," Proc IEEE 55th Vehicular Technology Conference, 2002, pp. 1160–1164.

[28] H.F. Salama, D.S. Reeves, and Y. Viniotis, Evaluation of multicast routing algorithms for real-time communication on high-speed networks, IEEE J Selected Areas Commun 15 (1997), 332–345.

[29] L. Schwiebert and R. Chintalapati, Improved fault recovery for core based trees, Comput Commun 23 (2000), 816–824.

[30] J. Song, S. Kim, M. Lee, H. Lee, and T. Suda, "Adaptive load distribution over multipath in MPLS networks," Proc IEEE ICC, 2003, pp. 233–237.

[31] R. Sriram, G. Manimaran, and C.S.R. Murthy, A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees, IEEE/ACM Trans Networking 7 (1999), 514–529.

[32] A. Striegel and G. Manimaran, A survey of QoS multicasting issues, IEEE Commun Mag 6 (2002), 82–87.

[33] Q. Sun and H. Langendoerfer, "Efficient multicast routing algorithm for delay-sensitive applications," Proc 2nd Int. Workshop on Protocols for Multimedia Systems (PROMS'95), 1995, pp. 452–458.

[34] Q. Sun and H. Langendoerfer, An efficient delay-constrained multicast routing algorithm, J High-Speed Networks 7 (1998), 43–55.

[35] H. Ural and K. Zhu, "Fault recovery for a distributed SP-based delay constrained multicast routing algorithm," Proc IEEE IPDPS'02, 2002, pp. 242–251.

[36] Y. Wang, Z. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," Proc IEEE INFOCOM, 2001, pp. 565–571.

[37] B.M. Waxman, Routing of multipoint connections, IEEE J Selected Areas Commun 6 (1988), 1617–1622.

[38] R. Widyono, The design and evaluation of routing algorithms for real-time channels, Technical Report TR-94-024, Tenet Group, University of California at Berkeley, 1994.

[39] B. Zhang, M.M. Krunz, and C. Chen, A fast delay-constrained multicast routing algorithm, Proc IEEE ICC, 2001, pp. 2676–2680.

[40] Q. Zhu, M. Parsa, and J.J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," Proc IEEE INFOCOM, 1995, pp. 377–385.