An Efficient QoS Routing Algorithm for Quorumcast Communication *

Bin Wang
Dept. of Computer Science and Engineering
The Wright State University
Dayton, OH, 45324
bwang@cs.wright.edu

Jennifer C. Hou
Dept. of Computer Science
University of Illinois at Urbana Champaign
Urbana, IL 61801

Abstract

This paper extends the concept of multicast to quorumcast, *a generalized form of multicast communication. The need of quorumcast communication arises in a number of distributed applications. Little work has been done on routing quorumcast messages. The objective of previous research was to construct a minimum cost tree spanning the source and the quorumcast group members. In this work, we further consider the path quality of a constructed spanning tree in terms of delay constraints required by applications that use the tree. As the delay-constrained quorumcast routing problem is NP-complete, we propose an efficient heuristic QoS routing algorithm. We also consider how a loop is detected and removed in the course of tree construction and how to deal with member join/leave in/from the quorumcast pool. Our simulation study shows that the proposed algorithm performs well and constructs a quorumcast tree whose cost is close to that of the "optimal" routing tree.*

1. Introduction

Multimedia applicationsare becoming increasingly popular. As an increasing number of these applications are distributed in nature, and these applications can benefit from using the multicast paradigm, multicast communication has received a great deal of attention. This is evidenced by the deployment of native multicast services over the Internet2.

In the multicast paradigm, messages are delivered from a source to a set of members in a multicast group. Multicast routing is usually realized by constructing a multicast routing tree that spans the source and the multicast group members, subject to some optimization objectives. Data generated by the source(s) flows through the multicast tree, traversing each tree edge exactly once. Therefore, multicast makes more efficient network bandwidth utilization. In this paper, we study a generalized form of multicast that we term as *quorumcast* communication. Instead of sending messages to all the members in a multicast group, sometimes it suffices to (and is preferable to) send messages to a subset of the group members, where the members in the subset are *not* known *a priori*, but are selected dynamically from the multicast group which we term as the *quorum pool* in the context of quorumcast communication.

The need of quorumcast communication arises in a number of distributed applications. For example, synchronized update of data items based on quorum consensus [1] in a distributed database shared by several processes. This can be achieved by having a process issue requests for votes to a subset of processes. Update proceeds only when a quorum is obtained. Another example is resource discovery in a networked environment. A request is sent to a subset of servers and a small number of servers that can best serve the request are selected by comparing the responses from the servers. A third area that may leverage the notion of quorumcast is Web applications. Web caching has been recognized as one of the effective schemes to alleviate service bottlenecks and minimize the user access latency. In order to minimize the cost of cache misses, a cache routing algorithm may use the quorumcast paradigm and route requests to one or more proxies (or Web servers) which are believed to contain the desired documents [21]. Quorumcast communication is also a generalization of *anycast* [24, 5, 8] where a message needs to be delivered to only one out of a group of members.

Emerging networks have been increasingly orchestrated to provide quality of service (QoS) to applications. This has been reflected in the networking industry and research community's on-going efforts to provision differentiated services. In the above distributed database synchronization problem, it is desirable for a process to receive responses from a subset of processes within a bounded delay so that the update operations can be timely performed. Similarly in resource discovery applications, the client can be better served if capable servers can be located within a predictable short amount of time. The same argument applies to Web applications.

In this paper, we study the problem of constructing quorumcast routing trees subject to certain QoS constraints. One objective (used in previous work [6, 2]) is to minimize the total communication cost, determined as the sum of the cost of the links in the routing tree. We further impose that the cumulative delay from the source to each of the members in the quorumcast group Q is within a pre-

*The work reported in this paper was supported in part by NSF under Grant No. ANI-9804993 and ITEC-Ohio. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

1092-1658/01 $17.00 © 2001 IEEE

specified delay bound. We term this routing problem as the *delay-constrained quorumcast routing* ($DCQR$) problem. As the problem is NP-complete, we present an efficient *distributed* QoS routing heuristic algorithm. We evaluate the proposed algorithm through simulation and compare it against the baseline exhaustive-search algorithm and two other existing algorithms. Our simulation study indicates the proposed algorithm constructs a quorumcast routing tree whose cost is close to that of the "optimal" tree (constructed by the exhaustive-search algorithm).

The rest of the paper is organized as follows. In Section 2, we introduce the network model, formulate the quorumcast routing problem, and articulate the differences between quorumcast and multicast. We then present the proposed QoS quorumcast routing algorithm in Section 3. The performance of the proposed algorithm is evaluated in Section 4 through simulations using randomly generated networks. The related work is summarized in Section 5. Finally, we conclude the paper with future work in Section 6.

2. Problem Description

In this section, the network model and the delay constrained quorumcast routing problem are presented.

2.1 Network Model

We represent a network by a weighted digraph $G = (V, E)$, where V denotes the set of nodes, and E the set of arcs. E corresponds to the set of network communication links connecting the nodes. We use $|V|$ to refer to the number of nodes in the network. Without loss of generality, we only consider graphs with at most one arc between a pair of ordered nodes. We define a link-delay function $f_D : E \rightarrow R^+$ which assigns a non-negative weight to each link in the network. The value $f_D(l)$ associated with link $l \in E$ is a measure of the delay that packets incur on link l, and takes into account the queueing delay, the transmission delay, and the propagation delay. We also define a link-cost function $f_C : E \rightarrow R^+$ which assigns a non-negative weight to each link in the network. The value $f_C(l)$ associated with link $l \in E$ is a measure of the cost for using link l, which can be, for example, a function of the link utilization. The link delay and cost are assumed to be collected and maintained by some underlying topology-update or routing protocol (e.g., link-state routing protocol) at each node in the network, and is updated periodically.

2.2 Problem

We generalize the multicast paradigm to the *quorumcast* paradigm. The former refers to distributing information to each member in a group while the latter refers to distributing information to (and preferably to) a subset of members in a group.

Let $M \subset V$ ($|M| = m$) be a set of nodes called a *quorum pool*. Let Q be a set of nodes ($|Q| = n$) whose members are to be dynamically selected (thus not known *a priori*) from the quorum pool M, i.e., $Q \subset M, n \leq m$. We call Q a quorumcast group. Let node s be a source node. Messages originating from node s are delivered to all members in Q. The routing problem is then to construct a quorumcast routing tree $T = (V_T, E_T^Q)$ (V_T and E_T^Q are the sets of nodes and links in T, respectively) which spans the source node s and all the nodes in Q. In addition, the tree T may include relay nodes, i.e., non-quorumcast group member nodes along the path from the source node to the destination nodes in Q. Our objective is to select a set of quorumcast nodes Q from the quorum pool M such that given a source node s, the cumulative delay along the path from s to any destination node $v_d \in Q$ satisfies a pre-specified delay constraint D, and the total cost of the routing tree is minimized. We term this routing problem as the *Delay Constrained Quorumcast Routing ($DCQR$)* problem. We formally state the $DCQR$ problem as follows:

Problem 1 *($DCQR$) Given a network $G = (V, E)$, a link-delay function $f_D : E \rightarrow R^+$ and a link-cost function $f_C : E \rightarrow R^+$, a source node s, a quorum pool M ($|M| = m$), and a delay bound D, select a subset of nodes, $Q \subset M$ ($|Q| = n, n \leq m$) and construct a routing tree $T = (V_T, E_T^Q)$ that is rooted at the source node s and spans the quorumcast group members in Q, such that:*

$$min_{\forall Q \subset M} \sum_{\forall l \in E_T^Q} f_C(l), \quad (1)$$

$$\textit{subject to} \sum_{\forall l \in P_{T(s,v_d)}} f_D(l) \leq D, \quad \forall v_d \in Q. \quad (2)$$

The $DCQR$ problem is a generalization of the delay constrained *Steiner tree* problem because when $Q = M$, the $DCQR$ problem reduces to the delay constrained *Steiner tree* problem. The latter is known to be *NP*-complete [20]. Therefore the $DCQR$ problem is also *NP*-complete.

The difference between multicast and quorumcast is subtle: the multicast group members spanned by a multicast routing tree is known *a priori* (namely M), while the quorumcast group members in Q spanned by a quorumcast routing tree is not known in advance. The quorumcast routing problem is therefore a generalization of the multicast routing problem.

3. Proposed Algorithm

A naive way of constructing an optimal delay constrained quorumcast routing tree is to select n nodes from the quorum pool M and then construct $\binom{m}{n}$ different delay constrained Steiner trees using known heuristics [20]. Among $\binom{m}{n}$ different routing trees thus constructed, the one that has the minimum cost is the optimal routing tree. Albeit simple, this naive algorithm incurs excessively high computational cost. In addition, almost all known heuristics are centralized and may introduce single points of failure.

3.1 Overview of the Proposed Algorithm

Instead of having each node maintain full knowledge of the network as required by centralized heuristics, we pro-

DestNode	MinDelay	NextHop

Figure 1. Fields for RTD table.

DestNode	MinCost	NextHop	MinDelayFromNextHop

Figure 2. Fields for RTC table.

pose a distributed algorithm. Our algorithm only uses information kept locally at each node. The construction of a routing tree starts from the source node s. Quorumcast group members are repeatedly selected and connected to the routing tree one at a time until n members are on tree. Each member is connected to a partial routing tree in such a way that the cumulative delay from the source node s to the member under consideration is less than or equal to the delay bound D. Moreover, the cost of the route used to connect this member to the partial tree is as low as possible.

Before delving into the details of our proposed algorithm, we first discuss the maintenance of the auxiliary routing information used by our algorithm. At each network node, we maintain two tables: one (RTD) that keeps the shortest delay path to every other node, and the other (RTC) that keeps the minimum cost path to every other node in the network. The RTD table consists of $|V|$ entries, one for each node in the network. Each entry in the RTD table has three fields (Fig. 1):

- *DestNode*: the destination node v;
- *MinDelay*: the delay of the shortest delay path from source node s to node v; and
- *NextHop*: the next hop neighbor on the shortest delay path to node v.

Similarly, the *RTC* table consists of $|V|$ entries, one for each node in the network. Each entry in the *RTC* table has four fields (Fig. 2):

- *DestNode*: the destination node v;
- *MinCost*: the cost of the minimum cost path from source node s to node v;
- *NextHop*: the next hop neighbor on the minimum cost path to node v; and
- *MinDelayFromNextHop*: the cumulative delay from the next hop neighbor *NextHop* to node v along the shortest delay path from source s to node v. This information can be obtained by letting neighboring nodes exchange delay information (i.e., $RTD[v].MinDelay$). Alternatively, each node may inquire its next hop neighbor along the path to node v for this information (i.e., $RTD[v].MinDelay$) on demand.

Notice that the information kept in the two tables is similar to the distance vectors kept by some existing routing protocols, e.g., RIP [12]. The table entries can be updated using the distributed Bellman-Ford's algorithm [7]. Moreover, the techniques used to deal with the convergence problem and the route instability problem in distance vector based routing protocols, such as split horizon, reverse poison, triggered route update, and hold-down of expired routes [14], can be readily applied to maintain the delay and cost information in the *RTD* table and the *RTC* table.

3.2 Detailed Description of the Routing Algorithm

Our distributed quorumcast routing algorithm works as follows (Fig. 3). Each node runs the same algorithm. The source node initiates the route construction process. The algorithm selects, among all the member nodes $\in M$ whose shortest delay paths from the source node s satisfy the delay constraint D, the one whose cost of the minimum cost path from the source node s to the node under consideration is the smallest. The algorithm then proceeds to find the best (in terms of cost) feasible route (i.e., a route that satisfies the delay bound) to the selected node and adds the node to the quorumcast group Q. In the following, we first describe the algorithm assuming that no loop will form. Then, we discuss situations in which loops may occur and devise a loop detection and removal procedure.

The source node s inspects each node $v \in M$ and looks up in the table RTD the delay of the shortest delay path from itself to node v. If $RTD[v].MinDelay > D$, then no path that satisfies the delay constraint (henceforth called the *delay-constrained* path) exists between s and v (v is then excluded from further consideration). If delay constrained paths do exist for some nodes in M, the algorithm selects the one whose minimum cost path incurs the lowest cost. Let this selected node be denoted as v'. The source node s sends a *setup* message along the least cost path towards node v'. The *setup* message keeps track of the cumulative delay along the path from the source node s to node v' in the corresponding *setup* message field D_u (which is set to 0 initially at the source node). Each intermediate node u, upon receipt of the *setup* message, records in its routing table (i) the neighbor node from which the *setup* message arrived (ii) the node which initiates the *setup* message, and (iii) the destination of the *setup* message. The node u then checks if

$$D_u + d(u, RTC[v'].NextHop) + \\ RTC[v'].MinDelayFromNextHop \le D,$$

where $d(u, RTC[v'].NextHop)$ is the one-hop delay from node u to node $RTC[v'].NextHop$. If the inequality holds, there exists a delay constrained path from the current node u to node v' that uses part of the minimum cost path. On the other hand, if the inequality does not hold, the algorithm selects the next hop $RTD[v'].NextHop$ along the shortest delay path to send the *setup* message. Node u records the identity of the next hop, the cumulative delay (D_u), and the direction (the minimum cost path or the shortest delay path) along which the *setup* message is forwarded. Node u then forwards the *setup* message to the next hop. When node v' receives the *setup* message, the tree branch from source node s to node v' is established.

Now node v' has to select the next node from $M - \{v'\}$. The node that has a feasible delay constrained path and whose minimum cost path from any node on the partial routing to the node under consideration incurs the smallest cost will be selected. A table $SELECTION$ is included in the *setup* message and initialized at the source s. Each member v in the quorum pool M has

```
1. Procedure QoS_Quorumcast_Routing()
2. {
3.   while receiving a message do
4.   switch(message.type) {
5.      case setup: setup(); break;
6.      case fork: fork(); break;
7.      case finish: finish(); break;
8.      case join: join(); break;
9.      case leave: leave(); break;
10.     case loop-removal: loop-removal();
11.  }/* switch */
12. }
13. Procedure setup ()
14. {
15.  if (self is the source node) { /* source initiates route setup */
16.     D_u = 0; initialize SELECTION;
17.     foreach (destination i in SELECTION) {
18.        SELECTION[i].cost = RTC[i].MinCost;
19.        SELECTION[i].OnTreeNode = s;
20.        SELECTION[i].tage = No; }
21.     Exclude from SELECTION, node w ∈ M and
22.     d_min(s, w) > D;
23.     Choose node v, s. t. cost(s, v) is lowest;
24.     SELECTION[v].tag = yes;
25.     message.dest = v;
26.  } /* if */
27.  else { /* if the node is not a source */
28.     update D_u at u;
29.     record message source and destination addresses at u;
30.     foreach (v' such that SELECTION[v'].tag == No) {
31.        if (D_u + d(u, RTC[v'].NextHop)+
32.            RTC[v'].MinDelayFromNextHop ≤ D and
33.            SELECTION[v'].cost > RTC[v'].MinCost) {
34.            SELECTION[v'].cost = RTC[v'].MinCost;
35.            SELECTION[v'].OnTreeNode = u;}
36.     }
37.  }/* else*/
38.  if (current node ≠ message.dest){ /* keep forwarding */
39.     message.type = setup;
40.     message.SELECTION = SELECTION;
41.     if (D_u + d(u, RTC[v].NextHop)+
42.        RTC[v].MinDelayFromNextHop ≤ D )
43.        forward message to lowest cost next hop RTC[v].NextHop;
44.     else
45.        forward message to shortest delay next hop
46.        RTD[v].NextHop;
47.  } /* if */
48.  else /* current node == message.dest*/
49.     if(|Q| members are on tree or all tags are yes)
50.        send finish to s;
51.     else {
52.        select v' s. t. SELECTION[v'].tag == no and
53.        SELECTION[v'].cost being the lowest;
54.        message.type = fork;
55.        message.SELECTION = SELECTION;
56.        send a fork message to SELECTION[v'].OnTreeNode;}
57. }
58. Procedure fork ()
59. {
60.  SELECTION = message.SELECTION;
61.  select v' with SELECTION[v'].tag == No and
62.  SELECTION[v'].cost being the lowest;
63.  SELECTION[v'].tag = yes; message.dest = v';
64.  message.SELECTION = SELECTION;
65.  message.type = setup;
66.  if (D_u + d(u, RTC[v'].NextHop)+
67.     RTC[v'].MinDelayFromNextHop ≤ D )
68.     forward message to lowest cost next hop RTC[v'].NextHop;
69.  else
70.     forward message to shortest delay next hop
71.     RTD[v'].NextHop;
72. }
```

Figure 3. The distributed QoS quorumcast routing algorithm.

```
73. Procedure loop-removal ()
74. {
75.  if (message source and destination match addresses
76.     previously recorded and setup used lowest cost
77.     next hop and has an alternate shortest delay next hop)
78.     send setup message to RTD[v].NextHop;
79.  else
80.     forward loop-removal backwards along the loop;
81. }
```

Figure 3. The distributed QoS quorumcast routing algorithm (Cont'd). Procedures *finish* and *leave* are not listed due to their simplicity. The *join* procedure is discussed in Section 3.4

an entry in table $SELECTION$ (except any node w with $RTD[w].MinDelay > D$). Each entry in the $SELECTION$ table has three fields:

- $SELECTION[v].cost$: the cost of the minimum cost path from the on-tree node $SELECTION[v]$ $.OnTreeNode$ to node v;
- $SELECTION[v].OnTreeNode$: the on-tree node at which node v will join the routing tree; and
- $SELECTION[v].tag$: a flag that indicates whether or not node v is on the routing tree.

The *OnTreeNode* and *tag* fields are initialized to s and "*no*," respectively.When a node v is selected, the *tag* field of its corresponding entry is marked as "*yes*." When a *setup* message arrives at an intermediate node, u, on the way to node v, for each $v'(\in M) \neq v$ that is not already on the partial routing tree, the entry $SELECTION[v']$ is updated if

$$D_u + d(u, RTC[v'].NextHop)+$$

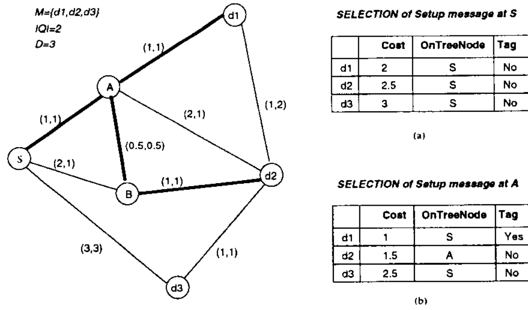
$$RTC[v'].MinDelayFromNextHop \leq D, \text{and}$$

$$SELECTION[v'].cost > RTC[v'].MinCost.$$

If both conditions are satisfied, then $SELECTION$ $[v'].cost = RTC[v'].MinCost$ and $SELECTION[v']$. $On\ TreeNode = u$, i.e., node u may be a better on-tree node from which a tree branch is grafted to node v'.

When the *setup* message reaches the selected node v', the next node v is selected as the node that has the smallest value of $SELECTION[v].cost$. The on-tree node recorded in $SELECTION[v].OnTreeNode$ will be the fork node at which a tree branch is grafted to node v. A *fork* message is then sent to this node. Upon receipt of the *fork* message, $SELECITION[v].OnTreeNode$ continues the tree construction process by sending a *setup* message toward v. $SELECTION[v].OnTreeNode$ is then included in the *setup* message as the source node. The above process continues until n members in M are on the tree or all tags *SELECTION* are marked as "*yes*" whichever occurs first. Finally, a *finish* message is sent to s. At this point, the quorumcast routing tree is constructed.

Example 1 *Fig. 4 gives an example that shows how a quorumcast routing tree is constructed.* $M = \{d_1, d_2, d_3\}$, $|Q| = 2$, *and* $D = 3$. *The routing tree construction starts from s. The initial* SELECTION *table is given in Fig. 4 (a). Node s initially selects* d_1 *as its first member to be added to the quorumcast routing tree, and sends a* setup *message to the next hop node A, along the minimum cost path. When*



SELECTION of Setup message at S

	Cost	OnTreeNode	Tag
d1	2	S	No
d2	2.5	S	No
d3	3	S	No

(a)

SELECTION of Setup message at A

	Cost	OnTreeNode	Tag
d1	1	S	Yes
d2	1.5	A	No
d3	2.5	S	No

(b)

Figure 4. An example that shows how a quorumcast routing tree is constructed. The two-tuple on a link is the $(cost, delay)$ pair. The *SELECTION* tables shown in (a) and (b) are the table carried in the *setup* message originating at source node s and that carried in the *setup* message at the intermediate node A, respectively.

the setup *message arrives at node A, the* SELECTION *table is updated. In particular, node A is found to be a better fork node at which node* d_2 *may join the routing tree. When the* setup *message reaches node* d_1, *node* d_2 *whose* $SELECTION[d].cost$ *is the smallest, is selected as the next member to join the routing tree. A* fork *message is sent to node A to notify it to send a* setup *message toward* d_2. *The process repeats and the constructed quorumcast routing tree is shown in bold lines.*

3.3 Loop Detection and Removal

In our algorithm, each node involved in the tree construction selects either the minimum cost path direction or the shortest delay path direction. If all nodes choose the minimum cost path direction, or the shortest delay path direction, then no loops occur. However, if some nodes choose the minimum cost path direction while others choose the shortest delay path direction, loops may occur (Fig. 5).

Loop detection is performed as follows. When a node receives a *setup* message, it searches its routing table. A loop is detected if a routing table entry already exists for the source-destination pair specified in the *setup* message. The current node that detects a loop initiates the loop removal operation by sending a *loop-removal* message to the previous node along the loop. The *loop-removal* message includes the *setup*-initiating node and the destination node. The *loop-removal* message traverses the loop backwards, removing routing table entries, until it reaches a node u whose routing table entry is marked as using the minimum cost path direction and which has an alternate shortest delay next hop. There must be at least one such node in the loop, otherwise a loop can not be formed. At this point, the *loop-removal* message is not sent further backwards. Node u then sends a *setup* message to $RTD[v].NextHop$ along the shortest delay direction, and updates its routing table to reflect the routing direction change. This decision can never lead to any delay constraint violation since the *setup* message is forwarded on the shortest delay path direction.

Example 2 *Fig. 5 shows an example in which a loop forms. The delay requirement is again 3. When* d_1 *is being connected, a* setup *message to* d_1 *arrives at A and A will forward the* setup *message to B along the minimum cost path direction because the cost of the minimum cost path via B*

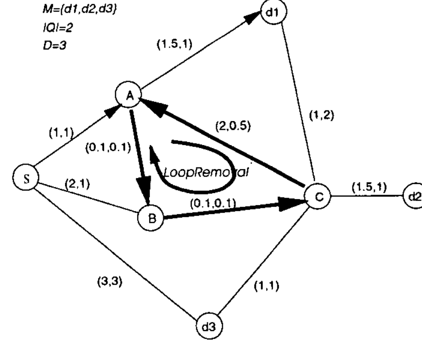


Figure 5. An example of loop formation, detection, and removal. The two-tuple on a link is the $(cost, delay)$ pair.

to d_1 *is 1.2* $(B \rightarrow C \rightarrow d_1)$ *and the cumulative delay from* s *to* d_1 $(s \rightarrow A \rightarrow B \rightarrow C \rightarrow A \rightarrow d_1)$ *is* $2.7 < 3$. *At node B, the* setup *message is forwarded along the minimum cost path to C, at which point the message has to be forwarded along the least delay path to A, because otherwise the cumulative delay is 3.2* $(s \rightarrow A \rightarrow B \rightarrow C \rightarrow d_1)$. *As a result, a loop forms. Node A detects the loop by comparing the source and destination addresses in the* setup *message against the entries stored in its routing table. A* loop-removal *message is then sent backwards along the loop until it reaches the first node that forwards the* setup *message along the minimum cost path and has an alternate shortest delay path, which in this example is node A. Node A then uses the shortest delay path to forward the* setup *message towards* d_1.

3.4 Dynamic Member Join and Leave

A node may dynamically become a member of the quorum pool M and a member in M may also leave. This happens, e.g., when an existing server fails, a new server is up and running, or an existing server becomes available again after a period of down-time. We adopt an approach that incrementally changes the quorumcast routing tree when a member joins/leaves M.

When a member v in M leaves, if this member v is in Q and is a leaf node of the quorumcast routing tree, it sends a *leave* message upstream with respect to the source node s. The *leave* message travels upstream along the on-tree branch until it reaches a fork node. Upon receipt of a *leave* message, an intermediate node (that is not a fork node) deletes its corresponding routing table entry. The rest of the quorumcast routing tree remains unchanged. On the other hand, if node v is not a leaf node, it is not disconnected from the quorumcast routing tree but remains as a message-relaying node. Leave of a member $v \in Q$ also triggers the source node s to select a replacement member from $M - Q$ to make up the member loss of Q. To reduce the service disruption to an ongoing quorumcast session, instead of tearing down the entire routing tree and constructing a new one, the procedure described in Section 3.2 is used.

The case in which a node v becomes a member in a quorum pool M is a little bit complicated. Recall that our objective is to construct a quorumcast routing tree that satisfies the delay constraint while minimizing the total cost of the

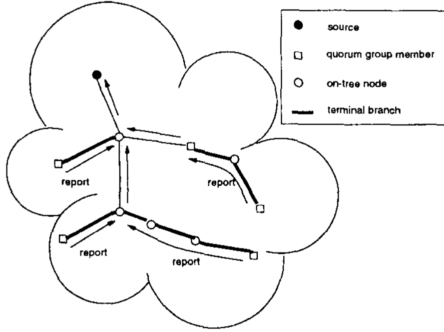


Figure 6. An example that shows the concept of a *terminal branch.*

tree. In order to minimize service disruption, one approach is simply to keep the tree unchanged till the next round of routing tree construction. At the other extreme of the solution spectrum, a new quorumcast routing tree can be constructed afresh, taking into account the new member. This will, however, result in temporary service disruption. To reduce the total tree cost while keeping the service disruption as minimum as possible, we propose a join procedure as follows.

We define a tree branch which emanates from an on-tree fork node, contains no more on-tree fork nodes, and leads to exactly one member in Q as a *terminal branch* (Fig. 6). When a node v becomes a member in M, it sends a *join* request to the source node s. The source node in turn multicasts a *join* message to current on-tree quorumcast members. The *join* message contains the cumulative delay D_u and the identity of node v. Upon receipt of a *join* message, an on-tree node w updates D_u. If w can locate (using the procedure listed in Fig. 3) a feasible delay constrained path that emanates from itself to node v and incurs a smaller cost than $SELECTION[v]$.*cost*, it updates the *cost* and *OnTreeNode* fields of $SELECTION[v]$. When a *join* message reaches a leaf node, the leaf node sends a *report* message back to the source node. A *report* message contains $SELECTION[v]$ and keeps track of the cumulative cost along the terminal branch towards its corresponding on-tree fork node. When a *report* message arrives at a terminal branch end-point, the cumulative cost along the terminal branch, the identity of the end-point, and $SELECTION[v]$ are forwarded to the source node. After receiving replies from all leaf nodes, the source node identifies (i) the least costly path that emanates from an on-tree fork node v_f and leads to the new M member and (ii) the most costly terminal branch and the corresponding terminal branch end-point v'_f. Item (i) is identified from all the received $SELECTION[v]$'s and item (ii) is determined from all the received cumulative costs along terminal branches. If the total tree cost can be reduced by a factor of δ when the most costly terminal branch is replaced with the least costly path leading to v, then the source node sends a *fork* message to the fork node v_f and a *prune* message is sent to the most costly terminal branch to trim the branch. Notice that the new tree branch grafted from node v_f to the new M member v may not eventually be the least costly path to v, because the least costly path may not satisfy the end-to-end delay requirement and at certain point, an intermediate node may have to switch from the least cost path to the shortest delay path. Therefore, the replacement does not always render a tree with a less cost, and should only be attempted when the cost difference between items (i) and (ii) is greater by a factor of δ. By properly setting the threshold δ, we can increase the likelihood that the resulting new routing tree will be less costly compared to the original one.

3.5 Complexity Analysis

Message overhead of constructing a routing tree: The complexity of the routing algorithm is evaluated in terms of the number of messages needed to construct a routing tree. It takes the following messages to construct a tree: (1) *setup* messages from source node s or fork nodes to $|Q|$ selected quorumcast group members. The longest possible path from a source node to a destination consists of $|V|$ nodes and $|V| - 1$ links. Therefore, the number of messages needed in the worst case for adding a member in Q is $O(|V|)$; (2) in the course of sending a *setup* message to a quorumcast group member, loops may form. In the worst case, a loop forms and the algorithm backtracks (with *loop-removal* messages) at each hop. It thus takes $O(|V|)$ messages to form/remove a loop, and $|V|^2$ backtracks in total may occur; (3) $(|Q| - 1)$ *fork* messages from $|Q| - 1$ selected group members to fork nodes; and (4) one *finish* message to inform the source node of the completion of the tree construction process. Thus, it takes $O(|Q| \cdot |V|) + O(|Q| \cdot |V|^2) + |Q| = O(|Q| \cdot |V|^2)$ messages to construct a routing tree to span $|Q|$ group members.

Message overhead of member join/leave: It takes $O(|V|)$ to tear down a tree branch in the case of a Q member leave and $O(|V|)$ messages to replace a member in Q. When a node becomes a member in M, it takes the following messages to replace a member in Q if necessary: (1) one message to notify the source node; (2) $O(|V|)$ *join* messages multicast from the source node and *report* messages from the leaving nodes to the source node since *join* and *report* messages are multicast over the existing routing tree; (3) $O(|V|)$ messages to add a new tree branch; and (4) $O(|V|)$ messages to tear down an old tree branch. Therefore, the number of messages needed in the worst case is $O(|V|)$.

Message overhead of information collection: Since the proposed approach relies on the underlying distance vector routing protocol to collect the information needed to update the auxiliary routing table, it does not introduce any additional message overhead.

4 Performance Evaluation

4.1 Network Topology Generation

The simulation study is conducted using a customized, Java-based network simulator, *NetSim*^{Q} [19]. The performance of the routing algorithm is evaluated using random network topologies generated using the method proposed by Waxman [22]. Sparse network topologies are generated

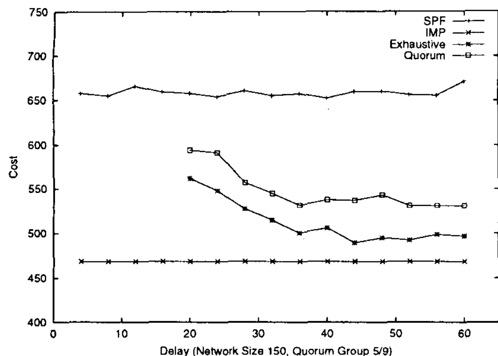


Figure 7. The tree cost versus the delay bound for the proposed algorithm, the exhaustive search approach, the SPT approach, and the IMP heuristic in a network of size 150 nodes with the 5/9 quorumcast group configuration.

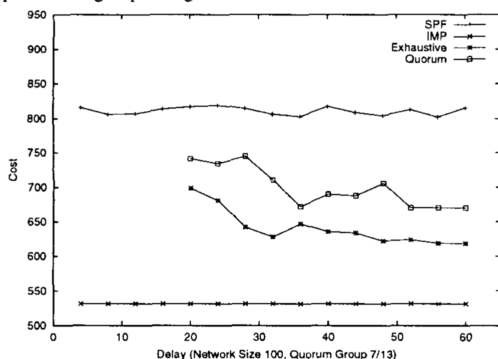


Figure 8. The tree cost versus the delay bound for the proposed algorithm, the exhaustive search approach, the SPT approach, and the IMP heuristic in a network of size 100 nodes with the 7/13 quorumcast group configuration.

with an average node degree 4. Each link is assigned (i) a delay which was chosen randomly from a uniform distribution $U(1.0, 10.0)$, and (ii) a cost which was chosen randomly from a uniform distribution $U(1.0, 100.0)$. Members of the quorum pool M are picked randomly from the set of nodes in the network. Each point in the performance graphs is an average over 200 simulation runs. Optimal quorumcast routing trees were obtained by an exhaustive search based on the dynamic programming technique.

4.2 Simulation Results

We deliberately consider small quorumcast groups as the applications considered in this work generally involve only a few group members. We use a/b to denote a quorumcast group size a and a quorum pool size b. We have conducted simulations in a wide range of configurations, we only report typical results obtained from simulation runs with $3/5, 5/9$, and $7/13$ (where the majority of members in the quorum pool M are in a quorumcast group).

In the first set of simulations, we compare the cost of the routing trees constructed by the proposed algorithm against that of the optimal tree and those constructed by two other approaches under various delay requirements for different

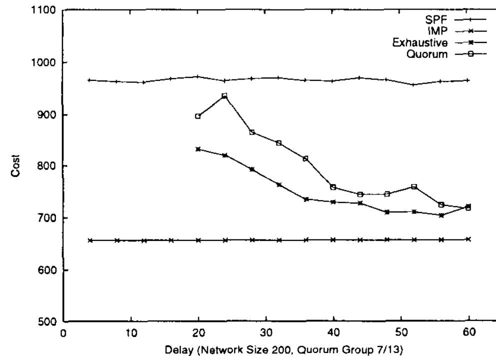


Figure 9. The tree cost versus the delay bound for the proposed algorithm, the exhaustive search approach, the SPT approach, and the IMP heuristic in a network of size 200 nodes with the 7/13 quorumcast group configuration.

network sizes and quorum group configurations. One approach obtains the quorumcast routing tree by choosing, among all the shortest path trees (SPT) that span a members from M, the one with the least cost. The other approach constructs the quorumcast tree using the IMP heuristic [6]. IMP is the best known heuristic for constructing a quorumcast tree *without delay requirements*. The network simulated is of size $25, 50, 75, 100, 150$, and 200 nodes. The configurations of the quorumcast groups and quorum pools are $3/5, 5/9$, and $7/13$, and the members in M are randomly selected.

Fig. 7 depicts the average tree cost versus the delay bound for a network size of 150 nodes and $5/9$ configuration. Since the SPT approach and the IMP heuristic are independent of delay requirements, their corresponding curves do not vary[1] with the delay bound. While the SPT approach strives to minimize the tree path delays without considering the tree cost, the IMP heuristic attempts to minimize the total tree cost regardless whether or not the resulting tree satisfies the delay constraint. Therefore, the SPT approach tends to construct trees with lower overall path delays but relatively higher costs, while the IMP heuristic renders trees with relatively lower costs without considering the delay requirements. Our proposed algorithm takes both delay and cost factors into consideration. When the delay bound is tight, satisfying the delay requirement has a higher priority over minimizing the tree cost. On the other hand, as the delay bound increases, the tree cost is reduced, which reflects the fact that the proposed algorithm attempts to develop tree branches along the least cost path in the course of constructing the tree.

Figs. 8-9 depict the tree cost versus different delay requirements in a network with the 7/13 configuration, and of size 100 and 200 nodes, respectively. Fig. 10 shows the tree cost versus different delay requirements in a network of size 200 nodes and with the 3/5 configuration. The total tree cost obtained using the proposed algorithm is consistently close to that obtained by the exhaustive search. In particu-

[1]The cost variation for the SPT approach is due to the fact that in the SPT approach, the minimum tree cost is approximated by the best routing tree obtained after fixed number of tries.

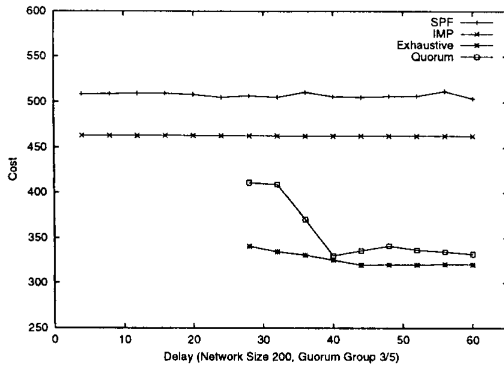


Figure 10. The tree cost versus the delay bound for the proposed algorithm, the exhaustive search approach, the SPT approach, and the IMP heuristic in a network of size 200 nodes with the $3/5$ quorumcast group configuration.

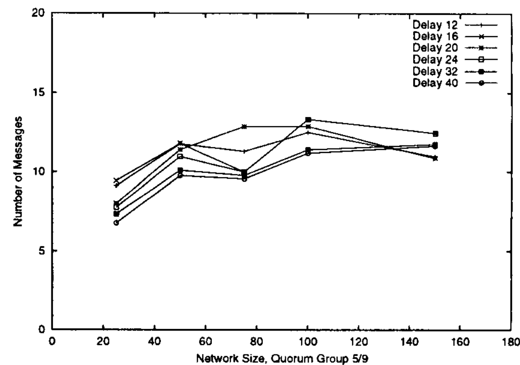


Figure 11. The message overhead versus the network size for the quorumcast configuration of $5/9$ with different delay bounds $12, 16, 20, 24, 32,$ and 40.

lar, for the case of very small group size ($3/5$) (Fig. 10), the IMP heuristic does not perform as well and the proposed algorithm outperforms the IMP heuristic.

In the second set of simulations, we study the message overhead introduced by the proposed algorithm. Fig. 11 gives the average number of messages versus the network size for different delay requirements: $12, 16, 20, 24, 32,$ and 40. As shown in Fig. 11, the average number of messages increases moderately with the increase of network size. When the delay bound is tight, the message overhead is small. This is because the proposed algorithm attempts alternately on the smallest delay path and the least cost path. When the delay bound is tight, the proposed algorithm is forced to use the smallest delay path most of the time, resulting in a SPT-like tree. This effectively reduces the likelihood of loop formation (and hence the number of *loop-removal* messages). When the delay bound increases, the number of control messages increases first. This is because the proposed algorithm has to weigh, and switch between, the two choices (the smallest delay path or the least cost path). This leads to an increase in the likelihood of loop formation, and hence contributes to the higher message overhead. As the delay bound increases further, the average number of control messages decreases, the proposed algorithm now tends to use the least cost path most of the time.

Finally, we investigate whether or not it is beneficial to reduce the total tree cost by trimming an existing tree branch and developing a new tree branch when a new member joins M. Our simulations indicate that due to the small group size ($3/5, 5/9, 7/13$), the possible locations to which a new branch could be grafted in order to replace an existing one are quite limited. Therefore, the probability of locating a new tree branch that simultaneously incurs lower cost and satisfies the delay requirement is very small. It turns out that it is not worthwhile to dynamically trim the tree when a new member joins M. One might as well construct a new quorumcast tree afresh at an appropriate time. In addition, for the type of applications we consider in this work, group members change only in rare circumstances.

4.3 Discussion

Compared with the centralized quorumcast algorithms in the literature [6, 2], our algorithm is distributed in the method of constructing a routing tree. It is efficient and incurs low message overhead. The information kept at each node is distance-vector-like information, not the detailed network topology and state information as required by many other existing algorithms. Each node keeps neither the network topology nor the states of individual links. The amount of information kept at each node is less than that required in centralized algorithms. Another major difference of using the proposed algorithm (as opposed to having the source perform all the calculation) is mainly its ability to utilize the information collected and "locally kept" by underlying unicast routing protocols to construct a tree in a decentralized manner. This avoids the overhead incurred in distributing the computation result from the source to other nodes. It also provides better fault tolerance and reduces the chance of using obsolete topology/cost/delay information at the source.

Our algorithm, however, has its disadvantages. Since members in Q joins the routing tree sequentially, the setup time may be long if $|Q|$ is large. We argue that for applications that may take advantage of quorumcast communication, the sizes of the M and Q are usually quite small and therefore the setup time will not be prohibitively large.

5 Related Work

Cheung *et al.* [6] studied the quorumcast routing problem with the objective of minimizing the total tree cost. They presented and evaluated several heuristics to find low cost routing trees. Ammar [2] proposed a probabilistic multicast paradigm in which a message is multicast to any probabilistically determined subset of the multicast group. The author explored use of this "probabilistic" multicast communication paradigm to address the scalability issues in response collection which arises in distributed computing applications. A lot of work has been focused on the Steiner tree problem that aims to minimize the total cost of a multicast tree [23, 10]. Unconstrained Steiner tree algorithms can

be used to solve this optimization problem. However, they do not attempt to fulfill other path constraints on an end-to-end basis, e.g., the end-to-end delay requirement from the source to any destinations, therefore may not be well-suited for applications with such requirements. Winter [23] and Hwang [10] did extensive surveys on both exact and heuristic Steiner tree algorithms. Bauer [3] and Salama [17] gave excellent reviews on most recent algorithms to the Steiner tree problem. The Steiner tree problem has been extended to include other side constraints, for example, delay, delay jitter, or a combination thereof. [9, 11, 16, 25]. Interested readers may refer to [20] for a detailed account. Our work differentiates from the previous work in that we adopt a distributed approach and in addition take into account the delay requirement in the quorumcast routing tree construction.

6 Concluding Remarks

In this paper, we have presented an efficient routing algorithm for a generalized form of multicast, *quorumcast* communication. In contrast to previous work on quorumcast routing, where the objective was to construct a minimum cost tree spanning the source and the quorumcast group members, we further consider the path quality of the constructed spanning trees in terms of delay constraints required by applications. Specifically, we impose a source-destination delay constraint. As the delay-constrained quorumcast routing problem is *NP*-complete, we propose an efficient heuristic QoS routing algorithm. We also consider how a loop is detected and removed in the course of tree construction and how to deal with member join/leave in/from the quorumcast pool. Our simulation study shows that the proposed algorithm performs well and constructs a quorumcast tree whose cost is close to that of the "optimal" routing tree.

We have identified several avenues for future research. First, the quality of a quorumcast routing tree may degrade over time as members in a quorumcast pool M dynamically join/leave, and a new quorumcast routing tree may have to be re-constructed at certain point. Bauer *et al.* [4], Sriram *et al.* [18], and Narvaez *et al.* [13] proposed several heuristics for locally rearranging part of an existing multicast tree so as to strike a balance between the multicast service disruption and the quality of a multicast tree. The suitability of leveraging these approaches in our proposed algorithm is an area of further study. Second, the soft state approach used in [15] can potentially be incorporated into our algorithm to enhance its robustness. This is also a direction for future research.

References

[1] M. Ahamad and M. H. Ammar. Performance characterization of quorum-consensus algorithms for replicated data. *IEEE Transactions on Software Engineering*, pages 492—496, April 1989.

[2] M. H. Ammar. Probabilistic multicast: Generalizing the multicast paradigm to improve scalability. *IEEE INFOCOM'94*, pages 848—855, 1994.

[3] F. Bauer. Multicast routing in point-to-point networks under constraints. *Ph.D. dissertation, University of California, Santa Cruz*, 1996.

[4] F. Bauer and A. Varma. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. *IEEE JSAC*, pages 382—397, April 1997.

[5] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, and Z. Fei. Application-layer anycasting. *Proceedings of INFCOM'97, Kobe, Japan*, 1997.

[6] S. Y. Cheung and A. Kumar. Efficient quorumcast routing algorithm. *Proc. of IEEE INFOCOM'94*, pages 840–847, 1994.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to algorithms. *MIT Press*, 1997.

[8] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar. A novel server selection technique for improving the response time of a replicated service. *IEEE INFOCOM'98*, March 1998.

[9] B. K. Haberman and G. Rouskas. Cost, delay, and delay variation conscious multicast routing. *Technical Report TR-97-03, North Carolina State University*, 1997.

[10] F. K. Hwang. Steiner tree problems. *Networks*, pages 55–89, 1992.

[11] V. P. Kompella, J. C. Pasquale, and G. C. Polyzo. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, pages 286–292, 1993.

[12] G. Malkin. RIP version 2: carrying additional information. *RFC 1723*, 1994.

[13] P. Narvaez, K. Siu, and H. Tzeng. New dynamic SPT algorithm based on a ball-and-string model. *IEEE INFOCOM'99, New York*, March 1999.

[14] T. Pusateri. Distance vector multicast routing protocol. *draft-ietf-idmr-dvmrp-v3-09.txt*, 1999.

[15] S. Raman and S. McCanne. A model, analysis, and protocol framework for soft state-based communication. *Proceedings of ACM SIGCOMM'99, Cambridge, USA*, September 1999.

[16] R. N. Rouskas and I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communications*, pages 346–356, April 1997.

[17] H. F. Salama. Multicast routing for real-time communication on high speed networks. *Ph.D. dissertation, North Carolina State University, Raleigh*, 1996.

[18] R. Sriram, G. Manimaran, and C. Murthy. A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees. *IEEE INFOCOM'99, New York*, March 1999.

[19] H.-Y. Tyan, B. Wang, and C.-J. Hou. NetSim^Q: A Java-integrated network simulation tool for QoS control in point-to-point high speed network s. *3rd NASA Research and Education Network Workshop, Moffett Field, CA*, August 1998.

[20] B. Wang and J. C. Hou. Multicast routing and its QoS extension: Problems, algorithms, and protocols. *IEEE Network*, 14(1):22–36, January/Feburary 2000.

[21] J. Wang. A survey of Web caching schemes for the Internet. *SIGCOMM Computer Communication Review*, October 1999.

[22] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, pages 1617–1622, December 1988.

[23] P. Winter. Steiner problem in networks: a survey. *Networks*, pages 129–167, 1987.

[24] E. W. Zegura, M. H. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: a server selection architecture and use in a replicated Web service. *IEEE/ACM Transactions on Networking*, August 2000.

[25] Q. Zhu, M. Parsa, and J. Garcia-Luna-Aceves. A source-based algorithm for delay-constrained minimal-cost multicasting. *Proc. of IEEE INFOCOM'95*, pages 377–384, 1995.