Journal of Parallel and Distributed Computing **61**, 137–149 (2001)
doi:10.1006/jpdc.2000.1668, available online at http://www.idealibrary.com on IDEAL®

Multicast Algorithms for Hypercube Multiprocessors

Shih-Hsien Sheu and Chang-Biau Yang

Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan 80424, Republic of China
E-mail: cbyang@cse.nsysu.edu.tw

Received August 17, 1995; revised February 8, 2000; accepted May 16, 2000

Depending on different switching technologies, the multicast communication problem has been formulated as three different graph theoretical problems: the Steiner tree problem, the multicast tree problem, and the multicast path problem. Our efforts in this paper are to reduce the communication traffic of multicast in hypercube multiprocessors. We propose three heuristic algorithms for the three problem models. Our multicast path algorithm is distributed, our Steiner tree algorithm is centralized, and our multicast tree algorithm is hybrid. Compared with the previous results by simulation, each of our heuristic algorithms improves the communication traffic in the corresponding multicast problem model.

Key Words: hypercube; multicast; NP-complete; heuristic algorithm.

1. INTRODUCTION

In a multiprocessor system, depending on the number of destinations, there are three types of interprocessor communications. They are *one-to-one* (*unicast*), *one-to-many* (*multicast*), and *one-to-all* (*broadcast*). Since unicast and broadcast communication in the hypercube have been studied intensively [4, 5], recently researchers have focused attention on multicast communication. A multicast communication in a network is where a source node wants to send a message to some destination nodes. The multicast is to determine which paths should be selected to deliver the message to the destination nodes such that the amount of communication traffic is reduced.

An n-cube graph is defined as $Q_n = (V, E)$, where $V = \{0, 1, 2, ..., 2^{n-1}\}$ and $E = \{(u, v) \mid$ the binary representations of u and v differ exactly on one bit position$\}$. For a multicast communication in Q_n, the input is a source node, denoted as u_0, and a multicast set of k destination nodes $M = \{u_1, u_2, ..., u_k\}$ to which we have to send the message from u_0. The *Hamming distance* between nodes u and v on Q_n,

0743-7315/01 $35.00

denoted as $H(u, v)$, is defined as the number of bit positions on which the binary representations of u and v differ.

The message transmission time is highly dependent on the underlying switching technology [9]. Eaxly direct networks used the *store-and-forward* switching. In this approach, the *network latency* is proportional to the distance, which is the number of hops (links) in the hypercube between the source and the destination nodes. Recently, more switching technologies have been proposed, such as *circuit switching*, *virtual cut-through*, and *wormhole routing* [9]. In these cases, the network latency or message transmission time is almost independent of the number of hops between the source and destination nodes. Depending on the underlying switching technology which a machine adopts, Lin and Ni formulated the multicast communication problem in multicomputers as three different graph problems [9]: the Steiner tree (ST) problem, the multicast tree (MT) problem, and the multicast path (MP) problem.

DEFINITION 1. Given a graph $G=(V, E)$, a source node u_0, and a multicast set $M \subseteq V$, a Steiner tree is a subtree $G_S=(V_S, E_s)$ of G such that $M \subseteq V_S$. A multicast tree is a subtree $G_T=(V_T, E_T)$ of G such that $M \subseteq V_T$ and for each $u \in M$, the path length from u_0 to u on T is equal to $H(u_0, u)$. A multicast path is a subgraph $G_P=(V_P, E_P)$ of G, where $V_P=\{v_1, v_2, ..., v_m\}$ and $E_P=\{(v_i, v_{i+1}) \mid 1 \leqslant i \leqslant m-1\}$, such that $u_0=v_1$ and $M \subseteq V_P$. The Steiner tree, multicast tree, and multicast path problems are to find the corresponding subgraph with minimum total length.

The optimization of all these three multicast problems on n-cube graphs has been proved to be NP-complete [3, 7, 9]. In this paper, we shall propose three heuristic algorithms to solve the three problems and compare each of them with the previous results [7, 9, 10] by some experiments. Our algorithms often need a smaller amount of communication traffic than the previous algorithms do. When implementing a multicast routing scheme, another issue is whether *centralized* routing or *distributed* routing should be adopted [7, 9, 10]. In this paper, our ST algorithm is centralized, our MP algorithm is distributed, and our MT algorithm is hybrid.

The rest of this paper is organized as follows. In Sections 2, 3, and 4, we shall propose our heuristic algorithms for solving the Steiner tree, multicast tree, and multicast path problems, respectively. We also give examples to illustrate our algorithms and previous algorithms in these three sections. In Section 5, we shall compare the performances of our multicast algorithms with the previous results on 10-cube by some simulation experiments. Finally, some conclusions will be given in Section 6.

2. A HEURISTIC ALGORITHM FOR ST

Lin and Ni [9] proposed an algorithm (we call it LinNi's ST algorithm) for constructing a Steiner tree on the hypercube. It consists of two phases: Phase 1; the message preparation phase, and Phase 2; the message routing phase. In Phase 1, they calculate the distance from the source to each destination node and then sort

the destination nodes in nondecreasing order according to the distances with the source node.

In Phase 2 (the message routing phase) of LinNi's algorithm, an important approach is that, for any three nodes s, t, and u in Q_n, a node v can be located in a constant time such that v is the nearest node to u among the nodes on all shortest paths from s to t. Let $u = u_{n-1}u_{n-2}...u_0$, $s = s_{n-1}s_{n-2}...s_0$, $t = t_{n-1}t_{n-2}...t_0$, and $v = v_{n-1}v_{n-2}...v_0$. Then v_j, $0 \leqslant j \leqslant n-1$, is set as $v_j = u_j$ if $s_j \neq t_j$ and $v_j = s_j$ if $s_j = t_j$. In Phase 2, initially, the source node and one of the destination nodes are linked as an initial Steiner tree. Then the Steiner tree is constructed by adding the destination nodes one by one in nondecreasing order. Note that each node in the Steiner tree is a node on Q_n and each edge in the Steiner tree corresponds to a path on Q_n. Let T denote the current Steiner tree. When a node u, which is not in T now, is going to be added into T, we will find the nearest node to u on each edge in T. After all the edges in T are examined, an edge (x, y) is chosen such that the distance between u and (x, y) is the minimum. Suppose that v is the node which is the nearest to u on (x, y). Then (x, y) is deleted and three new edges (x, v), (v, y), and (u, v) are added into T. A new partial Steiner tree is constructed. After all destination nodes are added, the final Steiner tree is obtained.

The final Steiner tree is deeply dependent on the order of adding the destination nodes into the tree. Thus, our effort is to focus on how to get a better order of adding the destination nodes. By the definition of the ST problem, a destination node may be reached via a nonshortest path. Adding the nearest destination node to the Steiner tree first may be helpless to minimize the total amount of traffic. Our main idea is to use the minimum spanning tree as the backbone of the Steiner tree. The nodes which are neighboring or near in the minimum spanning tree are often near in the Steiner tree. In our algorithm, we construct an auxiliary complete graph G consisting of the source and all destination nodes. For each edge (u, v) in G, the weight $w(u, v)$ is defined as $H(u, v)$. Then a minimum spanning tree T on G is found and a breadth-first search on T is performed to arrange the destination nodes in a breadth-first order. Finally, taking the destination nodes in the breadth-first order as the input of the message routing phase of LinNi's ST algorithm, we can obtain a Steiner tree. Our algorithm for solving the Steiner tree problem is as follows.

ALGORITHM STEINER TREE'ST

Input: Source node u_0 and multicast set M.

Output: A Steiner tree.

Step 1: Construct a complete graph $G = (V, E)$, where $V = \{u_0\} \cup M$, $E = \{(u, v) \mid u \in V \text{ and } v \in V\}$, and the weight of an edge $(u, v) = H(u, v)$.

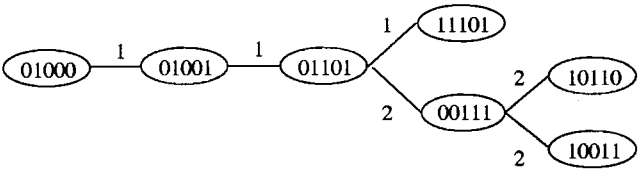
Step 2: Construct a minimum spanning tree T of G.

Step 3: Perform the breadth-first search on T, starting at u_0, and get a breadth-first order *BF*.

Step 4: Using *BF* as the input, perform the message routing phase of LinNi's ST algorithms. The output is a Steiner tree.

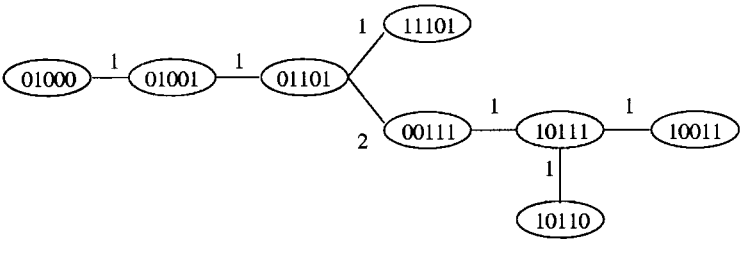
We give an example to illustrate how Algorithm ST works. Consider Q_5 in which node 01000(8) wants to send a message to each node in {10011(19), 11101(29), 01101(13), 01001(9), 10110(22), 00111(7)}. The minimum spanning tree T is shown in Fig. 1a and its breadth-first order is $BF = \{01000(8), 01001(9), 01101(13), 11101(29), 00111(7), 10110(22), 10011(19)\}$. Then we invoke the message routing phase of LinNi's ST algorithm, with BF as input, and a Steiner tree, as shown in Fig. 1b, is obtained. If we use the message preparation phase of LinNi's algorithm to arrange the order of the destination nodes, the ordered list is {01000(8), 01001(9), 01101(13), 11101(29), 10011(19), 10110(22), 00111(7)} and the resulting Steiner tree is shown in Fig. 1c. The communication traffic of the trees in Fig. 1b and Fig. 1c are 8 and 9, respectively.

Note that Algorithm ST is a centralized algorithm. Lin and Ni proved that the edges selected by their ST algorithm form a Steiner tree for the multicast set [9].

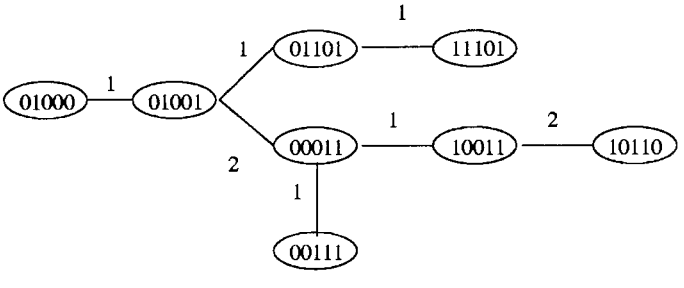


the breadth-first order: 01000, 01001, 01101, 11101, 00111, 10110, 10011.

(a)



(b)



(c)

FIG. 1. An example of a multicast Steiner tree. (a) A minimum spanning T and the breadth-first order for the destination nodes. (b) The Steiner tree obtained by our ST algorithm. (c) The Steiner tree obtained by LinNi's ST algorithm.

Thus, our approach also gets a Steiner tree for the multicast set. Suppose the Hamming distance $H(u, v)$ can be computed in a constant time. Step 1 of Algorithm ST takes $O(k^2)$ time, where $|M| = k$. If Prim's algorithm is used to find a minimum spanning tree, Step 2 takes $O(k^2)$ time. Performing a breadth-first search on T requires $O(k)$ time. At Step 4, every destination node has to check every edge of the Steiner tree. This step needs $O(k^2)$ time. Thus, the time complexity of Algorithm ST is $O(k^2)$.

3. A HEURISTIC ALGORITHM FOR MT

A famous distributed algorithm for the multicast tree problem on Q_n was proposed by Lan, Esfahanian, and Ni (LEN's MT algorithm) [7]. In their algorithm, when an intermediate node w receives the message and the multicast set M, it has to check if it is a destination node itself. If it is, it accepts the message locally and deletes itself from M. Then, w has to perform an XOR operation on its own binary address and the actual binary addresses of all elements in M to get their relative addresses. For a destination node u, the ith bit of u's relative address is 1 if u is different from w on dimension i. Hence, for each dimension i, $0 \leqslant i \leqslant n-1$, w counts how many elements there are in M whose ith bit of the relative address is 1. That is, if the count value of dimension i is c, there are c destination nodes whose addresses are different from w on dimension i. LEN's MT algorithm always chooses a particular dimension j with the maximum count value. All destination nodes whose relative addresses have 1 at bit j are sent to the jth neighbor of w. Then, these destination nodes are deleted from M. This procedure is repeated until the multicast set on w becomes empty.

Sheu and Su also proposed another MT algorithm [10], which is centralized, and the time complexity is $O(n2^n)$.

Since one of the goals in the MT problem is to minimize the traffic, a multicast tree is better than another if it has fewer tree nodes. Actually, for a 1-to-k multicast, at least $k+1$ nodes are needed to form a multicast tree. Suppose u and v are adjacent. The case in which we put these two nodes on the same path of the multicast tree will never be worse than that in which we put them on different paths, because the former case never leads to additional traffic. Based on this concept, in our approach, if two destination nodes are adjacent, we will try our best to put them in the same path.

Our MT algorithm consists of two phases, the neighbors linking phase and the message routing phase. The neighbors linking phase of our MT algorithm is executed only on the source node. It links the destination nodes in the multicast set which are adjacent. After this phase, the multicast set becomes a neighboring forest T and each element in T is a root of a tree. The routing phase of our MT algorithm is executed on each intermediate node in the multicast tree. This phase is similar to LEN's algorithm but a little different. First, if the local node w is an element in T, then w is deleted from T and the children of w are added to T as new elements. Second, for each dimension i, $0 \leqslant i \leqslant d-1$, we only count the nodes in T which are different from local node w on the ith dimension. Similarly, a dimension j with maximum count value is selected and each element in T whose address is different from

w on the jth dimension is sent to the jth neighbor of w with the message. Since each element v in T is a root of a tree, all elements which are the descendants of v are also sent with v. Then, all these elements are deleted from T on w. This procedure repeats until T on w is empty.

ALGORITHM MULTICAST TREE (MT)

The Neighbors Linking Phase

Input: Source node u_0, multicast set $M = \{u_1, u_2, ..., u_k\}$, and the dimension of the hypercube n.

Output: A neighboring forest T.

Step 1: Let $M_0 = \{u_0\}$.

Partition M into n disjoint sets $M_1, M_2, ..., M_n$, such that for each node v in M_i, $H(v, u_0) = i$. Note that M_i may be empty.

Step 2: For $i = n, n-1, ..., 1$ do

For each element $u \in M_i$ do

Check if u has an adjacent node in M_{i-1}

If so, say v, let v be the father of u and delete u from M_i.

Step 3: Let $T = M_1 \cup M_2 \cup \cdots \cup M_n$.

The Message Routing Phase

Input: Local node w and a neighboring forest T.

Goal: To build a multicast tree.

Step 1: If $w \in T$ then

Accept the message locally.
Let $T = T - \{w\} \cup \{v \mid v$ are the children of $w\}$.

Step 2: While $T \neq \varnothing$ do

(a) For $O \leqslant i \leqslant n-1$ do
count$[i]$ = number of elements in T that differ from w on dimension i.

(b) Let count$[j] = \max_{0 \leqslant i \leqslant n-1}\{count[i]\}$.

(c) $T' = \{v \mid v \in T, v$ and w differ on dimension $j\}$.

(d) Transmit T' and the message to node $(w \oplus 2^j)$.

(e) $T = T - T'$.

endwhile

Let us show an example to illustrate our algorithm. Consider Q_5. Suppose the source node is 00000(0) and the multicast set is $M = \{01010(10), 11101(29), 10001(17), 11111(31), 11100(28), 01011(11), 00010(2), 10110(22), 11110(30), 11011(27)\}$. Initially, in the source node, we partition M and obtain $M_1 = \{00010(2)\}$, $M_2 = \{01010(10), 10001(17)\}$, $M_3 = \{01011(11), 11100(28), 10110(22)\}$, $M_4 = \{11101(29), 11110(30), 11011(27)\}$, and $M_5 = \{11111(31)\}$. Note that in other examples, some of $M_i's$, $1 \leqslant i \leqslant n$, may be empty. We then link the

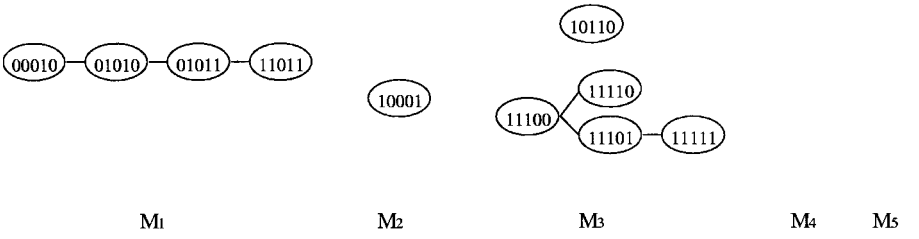


FIG. 2. An example of the neighboring forest in our MT algorithm.

neighboring nodes in different sets. The final result is $M_1 = \{00010\}$, $M_2 = \{10001\}$, $M_3 = \{10110, 11100\}$, $M_4 = M_5 = \varnothing$. Note that each element in M_i represents the root of a tree. The neighboring forest $T = M_1 \cup M_2 \cup M_3 \cup M_4 \cup M_5 = \{00010(2), 10001(17), 11100(28), 10110(22)\}$ is shown in Fig. 2. Then, the message routing phase is executed. The count value of each dimension in node 00000(0) is (3, 1, 2, 2, 1). Thus, $T' = \{10001(17), 11100(28), 10110(22)\}$ is transmitted to node 10000 with the message. Then $T' = \{00010(2)\}$ is transmitted to node 00010 with the message.

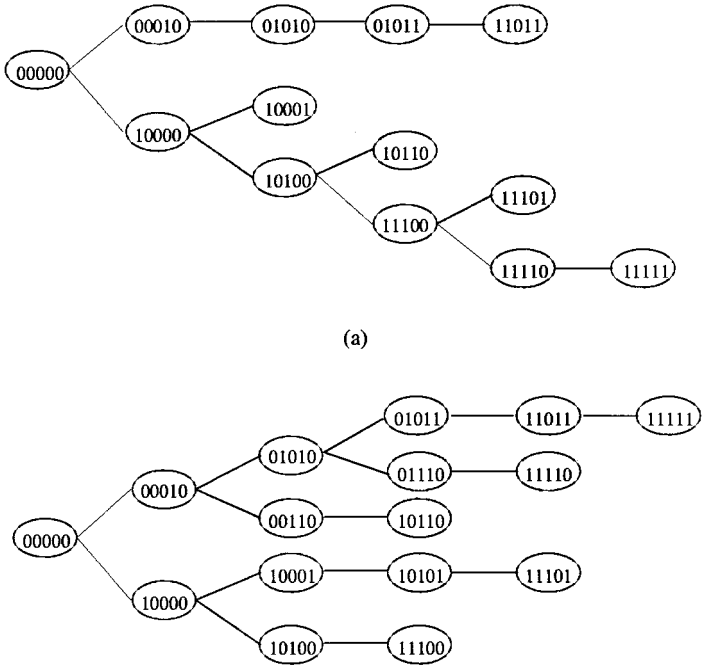


FIG. 3. Examples of multicast trees. (a) The multicast tree obtained by our MT algorithm, whose traffic is 12. (b) The multicast tree obtained by LEN's MT algorithm, whose traffic is 15.

Let us see the situation in node 11100. At this time, the neighboring forest is $T = \{11100(28)\}$. Since node 11100 is one of our destination nodes, it accepts the message locally. 11100 is deleted from T and 11110 and 11101, which are the children of 11100, are added into the forest T. Repeating this procedure in all intermediate nodes, we can obtain the final multicast tree, whose communication traffic is 12, as shown in Fig. 3a. The communication traffic of the multicast tree obtained by LEN's MT algorithm, as shown in Fig. 3b, is 15.

In Algorithm MT, the neighbors linking phase is executed only in the source node, and the message routing phase should be executed in each intermediate node. Thus, the algorithm is hybrid. In our approach, some far nodes are hidden behind a common ancestor until this ancestor is reached. Every destination node still receives a message via a shortest path from the source node. Except the neighbors linking phase of our algorithm, our approach is similar to LEN's MT algorithm. Since Lan *et al.* [7] have proved that given a source node and a multicast set, the edges selected by LEN's MT algorithm form a multicast tree, our algorithm also achieves this goal.

The time complexity of the neighbors linking phase is $O(nk)$ since Steps 1 and 2 need $O(k)$ and $O(nk)$ time, respectively, and the time complexity of the message routing phase is also $O(nk)$ since Steps 1 and 2 require $O(k)$ and $O(nk)$ time, respectively.

4. A HEURISTIC ALGORITHM FOR MP

Lin and Ni also proposed an algorithm (LinNi's algorithm) for constructing a multicast path for the hypercube topology [9]. Chen and Hwang modified LinNi's algorithm to be a balanced and deadlock free algorithm [1]. Both of them use the Hamiltonian path of Q_n as the backbone of the multicast path and cut off the unnecessary part of the Hamiltonian path.

The MP problem seems similar to the *traveling salesman* problem [2]. The well-known heuristic algorithm for solving the traveling salesman problem is the *nearest-neighbor* algorithm [2]. Since the destination nodes on Q_n cannot be visited in an arbitrary order, we do not really find a nearest node in our approach. In our approach, we always first visit the destination nodes in the same smallest subcube with the current node. Adding such nodes to the multicast path first will increase the amount of length less. In our algorithm, for a node u, we define $ld(u, w) = j$ if j is the leftmost bit position on which the binary addresses of u and the current local node w differ. For example, in Q_5, suppose the current local node is $w = 01011$ and there are two destination nodes $u_1 = 01101$ and $u_2 = 01010$. Then $ld(u_1, w) = 2$ and $ld(u_2, w) = 0$. In our algorithm, we always find the smallest ld function value, say s, among all destination nodes. Then we send the message with the multicast set to the sth neighbor of the current local node. Our method guarantees that a destination node which is in the smallest subcube with the current node will be the next target in our multicast path. Our heuristic algorithm, which is a distributed algorithm and is executed in each node in the multicast path, is as follows.

ALGORITHM MULTICAST PATH (MP)

Input: Local node w and a multicast set $M = \{u_1, u_2, ..., u_k\}$.

Goal: To find the next node to be added in the multicast path.

Step 1: If $w \in M$ then
w accepts the message locally.
Let $M = M - \{w\}$.

Step 2: If $M = \varnothing$ then stop.

Step 3: For each $u_i \in M$ do
$ld(u_i, w) =$ the leftmost different binary bit position between the addresses of u_i and w.

Step 4: Let $s = \min_{u_i \in M}\{ld(u_i, w)\}$.

Step 5: Transmit M and the message to node $(w \oplus 2^s)$.

The following theorem shows the correctness of Algorithm MP.

THEOREM 1. *Given a multicast set M in Q_n, the edges selected by Algorithm MP form a multicast path for M.*

Proof. We represent Q_n by a complete binar tree T_n of height $n+1$. Each node has two children. The values associated with the left and the right children are 0 and 1, respectively. The value of the root is undefined. Every path from the root to a leaf in T_n represents a binary address in Q_n. There are 2^n leaves in T_n, so T_n can represent all binary addresses in Q_n. In the following, when we mention a leaf, it is equivalent to the path from the root to that leaf, that is, it represents a binary address in Q_n. As an example, T_3 is shown in Fig. 4a, in which leaf u_1 represents 001. T_n shown in Fig. 4b represents Q_n.

Suppose that in our MT algorithm, the local node in Q_n corresponds to the leaf w in T_n. At Steps 3 and 4 of our algorithm, a destination node, say u_j, is found, where $ld(u_j, w) = s$ is the minimum among all destination nodes. Let node t be the ancestor of w on the $(s+1)$th level of T_n, where the level number is counted from 0 on the leaves. In addition, let T_l and T_r denote the subtrees rooted at the left and the right children of t, respectively. Without loss of generality, suppose w is in T_l. We claim that on any level $s+1$, once we leave T_l, there is no destination node in T_l which has not been visited yet, and T_l will never be entered from T_r again. It is clear that the claim is equivalent to this theorem.

Now, we prove the above claim. In each of the subtrees T_l and T_r, there are 2^s leaves. The leaves in T_l and T_r represent the binary addresses $b_{n-1}b_{n-2}\cdots b_{s+1}0\underbrace{XX\ldots X}_{s}$ and $b_{n-1}b_{n-2}\ldots b_{s+1}1\underbrace{XX\ldots X}_{s}$, respectively, where X may be 0 or 1. Note that T_l corresponds to a subcube Q_s and T_r corresponds to another subcube Q_s. If leaf nodes u, v are located in the same subtree T_l or T_r, the value of $ld(u, v)$ must be in the range from 0 to $s-1$. Nodes u, v are located in different subtrees, one in T_l and the other in T_r, if and only if the value of $ld(u, v)$ is s. Hence, if we want to transmit from T_l to T_r, the minimum value of an ld function must be s. Thus, there is no destination node in subtree T_l which has not been visited yet.

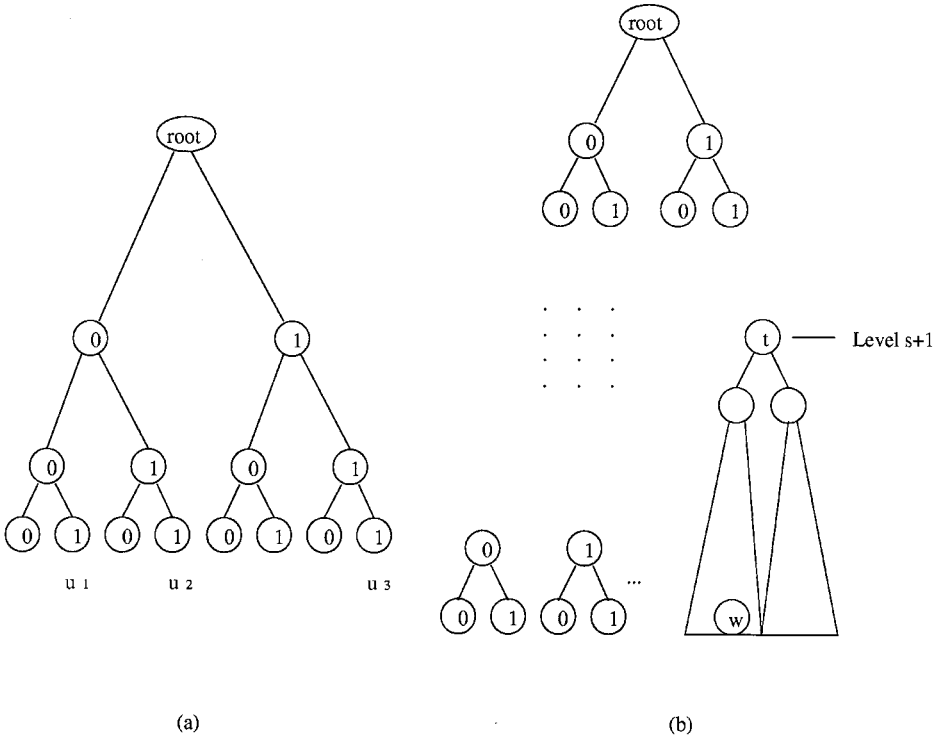


FIG. 4. Representing Q_n by a complete binary tree T_n: (a) T_3, (b) T_n.

After transmitting from T_l to T_r, since there is no destination node in T_l, it is impossible to find an *ld* function value that is equal to s. That is, after transmitting from T_l to T_r, we will never enter T_l from T_r. ▌

Now, let us show an example to illustrate how our MP algorithm works. Consider Q_6 in which the source node is $w = 110101(53)$ and the multicast set is $M = \{101011(43), 000100(4), 111001(57), 111010(58), 000001(1), 100001(33)\}$. The *ld* function values in the source node are 4, 5, 3, 3, 5, and 4, respectively. Then $\min_{u_i \in M}\{ld(u_i, w)\} = 3$.

Applying the algorithm, we obtain the multicast path $\{110101(53), 111101(61), 111001(57), 111011(59), 111010(58), 101010(42), 101011(43), 100011(35), 100001(33), 000001(1), 000101(5), 000100(4)\}$, whose communication traffic is 11. If the instance of this example is solved by LEN's MP algorithm, the final multicast path is $\{53, 61, 57, 59, 58, 56, 40, 41, 43, 35, 33, 1, 5, 4\}$, whose communication traffic is 13.

Assuming that the *ld* function can be computed in a constant time, Steps 3 and 4 of our MT algorithm spend $O(k)$ time. Therefore, the complexity of our MT algorithm is $O(k)$.

5. PERFORMANCE ANALYSIS

In this section, we shall illustrate the performance of our multicast algorithms on Q_{10} by simulation experiments. Table I is a summary of the complexities and

TABLE I

Complexities and Properties of the Multicast Algorithms

Problem model	Algorothms	Complexity	Property
The ST problem	LinNi's	$O(k^2)$	Centralized
	ours	$O(k^2)$	Centralized
The MT problem	LEN's	$O(nk)$	Distributed
	ours	$O(nk)$	Hybrid
	Sheu's	$O(n2^n)$	Centralized
The MP problem	LinNi's	$O(n \log n)$	Hybrid
	ours	$O(n)$	Distributed

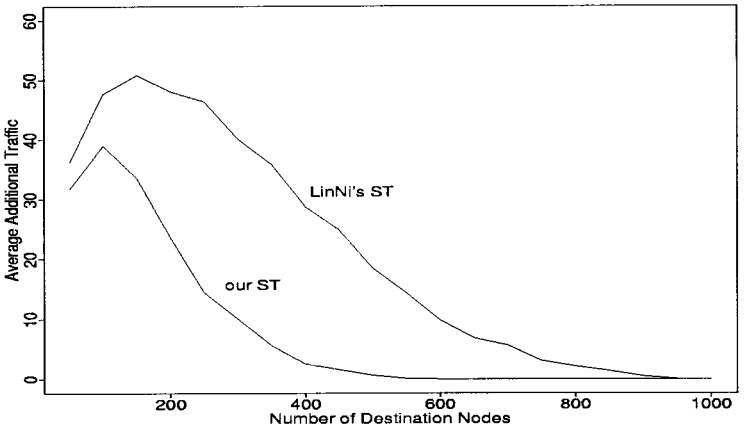


FIG. 5. Performance of Algorithm ST on Q_{10}.

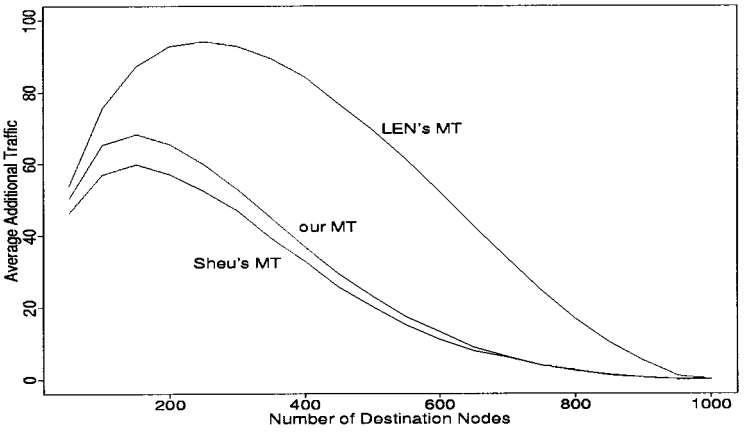


FIG. 6. Performance of Algorithm MT on Q_{10}.

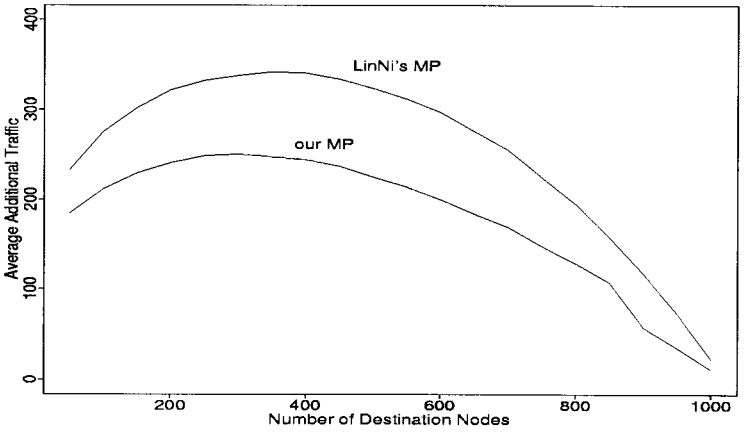


FIG. 7. Performance of Algorithm MP on Q_{10}.

properties (distributed, centralized, or hybrid) of the algorithms we mentioned in this paper. In general, the distribution of the destination nodes has a great effect on the total traffic generated by a multicast communication. But, similar to the performance study in most papers [7, 9, 10], we assume that the routing distribution is *uniform*, where the probabilities of sending the source message from the source node to nodes u_i and u_j are equal for all $u_i \neq u_j$. For a 1-to-k multicast, it requires at least k units of traffic, where a unit of traffic is measured as one message transmitted over one link. As in the previous studies [9, 10], we use *average additional traffic* to evaluate the performance of a multicast communication, where the average additional traffic is defined as the average amount of total traffic minus k. Figures 5, 6, and 7 show the amount of average additional traffic generated by our ST, MT, and MP algorithms, respectively. Each is compared with the previous algorithms. The number of destination nodes, k, is ranged from 50 to 1000 steps by 50. For each k, we perform the simulation 500 times and the amount of traffic generated for a given k is averaged over the 500 runs.

Each of our algorithms has a significant improvement on the corresponding previous algorithm except that the performance of Sheu's MT algorithm is slightly better than that of our MT algorithm. However, Sheu's MT algorithm is centralized while our MT algorithm is hybrid.

6. CONCLUSIONS

Based on different underlying switching technologies, Lin and Ni [9] defined three multicast problems: the Steiner tree problem, the multicast tree problem, and the multicast problem. Since all the three multicast problems on the hypercube were proved to be NP-complete [3, 7, 9], the previous algorithms for solving these problems are all heuristic. In this paper, we propose three new heuristic algorithms for solving them on the hypercube topology. By simulation experiments, our algorithms have significant improvements on the previous algorithms.

Fault-tolerance and deadlock are two issues that require further investigation. Several fault-tolerant algorithms were proposed for the MT problem [6, 8]. Lan's

fault-tolerant MT algorithm [6] is close to LEN's MT algorithm. Since our MT algorithm preserves the main idea of LEN's MT algorithm, it is possible to modify our MT algorithm to be fault-tolerant. In the switching technology of wormhole routing, several multicasts may compete for the common resources such as the communication channels. When there is a cycle in the channel dependence graph, deadlock may occur. Finding a way to modify our MP algorithm to prevent deadlock may be worth further studying.

REFERENCES

1. H. L. Chen and G. J. Hwang, An effective multicast wormhole routing in hypercube, *Internat. J. Electron.* **77**(1) (1994), 1–15.
2. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to Algorithms," first ed., MIT Press, Cambridge, MA, 1989.
3. R. L. Graham and L. R. Foulds, Unlikelihood that minimal phylogenies for realistic biological study can be constructed in reasonable computational time, *Math. Biosci.* **60** (1982), 133–142.
4. S. L. Johnson and C. T. Ho, Optimal broadcasting and personalized communication in hypercubes, *IEEE Trans. Comput.* **38**(9) (Sept. 1989), 1249–1268.
5. J. C. Konig and D. Sotteau, Symmetric routings of the hypercube, *Discrete Math.* **121** (1993), 123–134.
6. Y. Lan, Adaptive fault-tolerant multicast in hypercube multicomputers, *J. Parallel Distrib. Comput.* **23** (1994), 80–93.
7. Y. Lan, A. H. Esfahanian, and L. M. Ni, Multicast in hypercube multiprocessors, *J. Parallel Distrib. Comput.* **8** (1990), 30–41.
8. A. C. Liang, S. Bhattacharya, and W. T. Tsai, Fault-tolerant multicasting on hypercubes, *J. Parallel Distrib. Comput.* **23** (1994), 418–428.
9. X. Lin and L. M. Ni, Multicast communication in multicomputer networks, *IEEE Trans. Parallel Distrib. Systems* **4**(10) (1993), 1105–1117.
10. J. P. Sheu and M. Y. Su, A multicast algorithm for hypercube multiprocessors, *Parallel Algorithms Appl.* **2** (1994), 277–290.