

Short Paper

Multi-Node Broadcasting in Hypercubes and Star Graphs¹

YU-CHEE TSENG

*Department of Computer Science and Information Engineering
National Central University
Chungli, Taiwan 320, R.O.C.
E-mail: yctseng@csie.ncu.edu.tw*

In a hypercube or a star graph, given an unknown number of nodes located at unknown positions each intending to broadcast a message, we propose an efficient routing algorithm to solve this problem using asymptotically optimal or near-optimal transmission time.

Keywords: collective communication, hypercube, interconnection network, many-to-all roadcasting, parallel architecture, routing, star graph.

1. INTRODUCTION

Technological advances have made building complicated large-scale multi-computer networks possible. The binary hypercube [7] is undoubtedly one of the most popular interconnection networks for parallel processing. Another candidate is the star graph [1], which has been recognized as an attractive alternative to the hypercube. A large number of researches have studied the star graph, focusing on its topological properties [3, 11], embedding capability [6, 13], and communication capability [2, 8, 10, 14].

Given an interconnection network, one important issue is how to perform broadcasting, such as one-to-all broadcasting and all-to-all broadcasting. In this paper, we take one further step and study the *multi-node broadcasting* (or *many-to-all broadcasting*) problem in hypercubes and star graphs, where there is an arbitrary set of nodes in the network, each intending to broadcast a message to the rest of the network. This generalizes the one-to-all and all-to-all broadcasting problems. These broadcasting operations fall within the realm of *collective communication* in interconnection networks and can be found in many applications, such as parallel graph algorithms, parallel matrix algorithms, fast Fourier transformation, parallel compilers, cache coherence, etc.

In the multi-node broadcasting problem under consideration here, it is not known in advance how many nodes intend to perform the broadcast operation.

Received October 22, 1997; revised May 11, 1998; accepted June 8, 1998.

Communicated by Wen-Lien Hsu.

¹ This research was supported in part by the National Science Council of the Republic of China under Grant No. NSC87-2213-E-008-012 and Grant No. NSC87-2213-E-008-016.

Thus, the locations of these source nodes are not known. This problem, at first glance, seems to be fairly difficult due to the combinatorial nature of the locations of the source nodes. In this paper, we propose an efficient routing algorithm which uses asymptotically optimal or near-optimal transmission time. When the broadcasted messages are sufficiently large, the transmission time is provably within 2 or 8 times the optimum. Interestingly, the number of source nodes can be calculated on-line in the algorithm with only a small constant extra communication delay, and the achieved performance is insensitive to the locations of these source nodes.

The main technique used in this paper is based on the existence of $O(n)$ edge-disjoint spanning trees in an n -cube and n -star. The idea of using those spanning trees for one-to-all and all-to-all broadcasting in hypercubes was first proposed by [5]. Later similar results were established by [4, 14] for star graphs. An algorithm for the multi-node broadcasting problem on hypercubes was proposed in [12]. In [12], a global *prefix sum* was first executed to calculate the number of source nodes in the network. On the contrary, in this paper, no prefix sum is needed — our scheme can calculate the number of sources on-line during the propagation of broadcast messages. Furthermore, the load (i.e., traffic) on the $O(n)$ edge-disjoint spanning trees is more balanced than that incurred by the scheme presented in [12]. According to our simulation (see Section 4), our results are better than those in [12] for multi-node broadcasting on hypercubes under certain circumstances. Our scheme can even be generalized to solve a more general multi-node broadcasting problem in which each source node has a message of different length to be broadcast. (See Section 5 for more discussion.)

The rest of this paper is organized as follows. Preliminaries are given in Section 2. Our multi-node broadcasting algorithm is presented in Section 3. Performance analysis is given in Section 4. Conclusions are drawn in Section 5.

2. PRELIMINARIES

An n -dimensional binary hypercube, or simply an n -cube, is an undirected graph consisting of 2^n nodes each labeled by a distinct binary number $b_1b_2\dots b_n$ from 0 to $2^n - 1$. Two nodes which differ in exactly one bit are connected by an edge. An n -dimensional star graph, or simply an n -star, is an undirected graph consisting of $n!$ nodes. Each node is assigned a unique label $x_0x_1\dots x_{n-1}$, which is a permutation of n symbols $\{0, 1, \dots, n-1\}$. Given any node label $x_0\dots x_j\dots x_{n-1}$, let function g_i , $1 \leq i \leq n-1$, be such that $g_i(x_0\dots x_j\dots x_{n-1}) = x_j\dots x_0\dots x_{n-1}$ (i.e., swap x_0 and x_j and keep the rest of the symbols unchanged). In S_n , for any node x , there is an edge joining x and node $g_i(x)$. In this paper, each link (edge) is assumed to have full-duplex capability in that it can send data in both directions concurrently. More properties about hypercubes and star graphs can be found in many references cited in this paper.

In the *multi-node broadcasting* problem, there is an unknown number of s source nodes located at unknown positions, each intending to broadcast a message m bytes in size to the rest of the network. The network is assumed to be synchronous in that all nodes share the same physical clock. Switching technology follows the store-and-forward model; thus, sending a message of i bytes along a link

will take $T_s + iT_c$ time, where T_s is the *startup time* (to initiate the communication) and T_c is the *transmission time* (to deliver one byte of data along a link). We assume the use of the *all-port* model, so a node can simultaneously send and receive messages along all input and output links, respectively.

Lemma 1: *To perform multi-node broadcasting in an n -cube with s source nodes, a lower bound on the communication latency is*

$$\max \left\{ nT_s, \frac{sm(2^n - 1)}{2^n n} T_c \right\} \approx \max \left\{ nT_s, \frac{sm}{n} T_c \right\}. \tag{1}$$

To perform multi-node broadcasting in an n -star with s source nodes, a lower bound is

$$\max \left\{ \left\lfloor \frac{3(n-1)}{2} \right\rfloor T_s, \frac{sm(n! - 1)}{(n-1)n!} T_c \right\} \approx \max \left\{ \left\lfloor \frac{3(n-1)}{2} \right\rfloor T_s, \frac{sm}{n-1} T_c \right\}. \tag{2}$$

Proof: For both bounds, the first term is the product of the diameter and T_s , which comes from the necessity for a message to reach the farthest node. The second term is obtained by dividing the required network bandwidth by the total offered network bandwidth. \square

We call a directed tree an *in-tree* if the root node is the only node having an out-degree of 0 and all the other nodes have an out-degree of exactly 1. Intuitively, an in-tree is one in which every edge is pointing, directly or indirectly, toward the root node. (This can be easily proved by first observing the nodes at a distance of one from the root, then those at a distance of two from the root, then those at a distance of three, etc.) Similarly, an *out-tree* is one that can be obtained from an in-tree by reversing the direction of each edge of the in-tree.

We need two lemmas that have been established earlier.

Lemma 2 [5]: *In an n -cube, there exist n edge-disjoint out-trees, each located at a distinct root and having a height of n .*

Proof: These are the n edge-disjoint spanning binomial out-trees (each rooted at one neighbor of node 00...0) as defined in [5]. \square

Lemma 3: [4] *In an n -star, there exist $n - 1$ edge-disjoint out-trees, each located at a distinct root and having a height of at most $\left\lfloor \frac{3(n-1)}{2} \right\rfloor + 3$.*

Proof: In [4], it was shown that there exist $n - 1$ edge-disjoint out-trees all rooted at node 01...(n - 1) and having heights of at most $\left\lfloor \frac{3(n-1)}{2} \right\rfloor + 4$. Further, the root of the i -th tree, $i = 1..n - 1$, has only one out-going edge, which leads to node

$g_i(01\dots(n-1))$. Reversing the direction of this edge gives an out-tree rooted at $g_i(01\dots(n-1))$ with a height of at most $\left\lfloor \frac{3(n-1)}{2} \right\rfloor + 3$. Doing this for all trees would prove this lemma.

3. THE MULTI-NODE BROADCASTING ALGORITHM

Given a multi-node broadcasting problem with an unknown number of s source nodes located at unknown positions each intending to broadcast a message of size m bytes, the basic idea of our algorithm is as follows. For an n -cube or n -star, let T_1, T_2, \dots, T_t be the edge-disjoint out-trees in the network as described earlier ($t = n$ in case of an n -cube and $t = n - 1$ in case of an n -star). Also, let $\hat{T}_i, i = 1..t$, be the in-tree obtained from T_i by reversing the direction of each edge in T_i . Clearly, the t in-trees $\hat{T}_1, \hat{T}_2, \dots, \hat{T}_t$ are also edge-disjoint. At the beginning of the algorithm, each of the s source nodes will partition its broadcast message evenly into t sub-messages (so there are st sub-messages in the system). For each source node, it then sends each of its t sub-messages through one of the t in-trees $\hat{T}_1, \hat{T}_2, \dots, \hat{T}_t$, one sub-message per in-tree, toward the corresponding roots. After some time, the root of each $\hat{T}_i, i = 1..t$, will have collected s sub-messages from all s source nodes. Then, “simultaneously,” every root will start broadcasting the received s sub-messages through the out-tree T_i to all other nodes.

The above description is necessarily abstract. In our algorithm, a sub-message may be further partitioned into smaller packets, which are to be sent in a pipeline manner. This will become clear later. Another hidden problem in the above description is that the value of s is, in fact, unknown to each root. This raises one question: how do the roots of T_1, T_2, \dots, T_t synchronize with each other to “simultaneously” start broadcasting their sub-messages. In our scheme, *no* specific synchronization is performed; only some special messages, which incur a constant communication delay, are sent for this purpose.

Next, we will present our algorithm in more detail. The algorithm involves an integer parameter p , which is to be determined later for optimization purposes. Messages are packetized into $\frac{m}{tp}$ bytes per packet. Time is slotted into a length of $T_s + \frac{m}{tp} T_c$, so in each time slot, one packet can be delivered. There are two types of packets: data packets (for carrying broadcast messages) and *MARKER* packets (for control purposes). Every node should start executing the following steps at the same time.

Step 1: Each source node partitions its broadcast message evenly into tp packets, each containing $\frac{m}{tp}$ bytes (for ease of presentation, we assume that m is divisible by tp). Starting from the first time slot, the source node injects these packets *evenly* into the t in-trees $\hat{T}_1, \hat{T}_2, \dots, \hat{T}_t$ (i.e., p packets per tree). Any node receiving a data packet should help propagate it along the same in-tree where it was first injected until the tree’s root is reached.

- Step 2:** In each \hat{T}_i , $i = 1..t$, each terminal node² injects a *MARKER* packet into \hat{T}_i . For a terminal node which is not a source node, it should do so in the first time slot. For a terminal node which is also a source, it should do so in the $(p + 1)$ -th time slot (i.e., right after it has sent out its own data packets). A non-terminal node in \hat{T}_i should forward a *MARKER* packet to its parent in \hat{T}_i immediately after the following conditions become true: (i) it has collected one *MARKER* from each of its children in \hat{T}_i , and (ii) all data packets it received from its children in \hat{T}_i have been forwarded to its parent. This is recursively performed until the *MARKER* reaches the root of \hat{T}_i .
- Step 3:** In each \hat{T}_i , after the root node has received one *MARKER* from each of its children, it calculates the number of data packets it has received (which must be sp). Let $\beta = sp + h + 1$, where h is the height of \hat{T}_i (note that h is the same for all \hat{T}_i). Then, at the β -th time slot, the root of out-tree T_i starts to broadcast all sp data packets it has received to the rest of the network through T_i . All nodes in T_i should make copies of these packets and further forward them to their children, if any.

4. CORRECTNESS AND PERFORMANCE

Clearly, by step 1, all data packets will eventually arrive at their roots. Although the value of sp is unknown in advance, a *MARKER* packet serves as a delimiter to the root and indicates the end of the data packets in the corresponding sub-tree from which the *MARKER* is received. Thus, once a root node has collected a *MARKER* from each child, a correct value of sp can be found. (Note that there may still be some packets destined for other roots passing by.)

The β used in step 3 serves as an upper bound on the time slots, thus ensuring that all roots have received *MARKERS* from all their children. As there are sp packets destined for each root, and as in each time slot each link of an in-tree is able to deliver one packet, after $sp + h - 1$ time slots, all data packets will have arrived at their roots, where $h - 1$ is due to the pipelining effects. With one more time slot, all *MARKERS* will also arrive at their roots. (Again this is due to the pipelining effects.) Thus, after $sp + h = \beta - 1$ time slots, the network will be clear of any packets. It is for this reason that our scheme only incurs a constant (one time slot) extra communication delay in calculating the value of s , whereas any algorithm, in solely calculating this value, must incur an $\Omega(n)$ communication delay.

Finally, in step 3, each root of the out-tree T_i , $i = 1..t$, broadcasts sp packets through T_i . In T_i , the first packet will arrive at the farthest node within h time slots, after which time one more packet will arrive at each time slot. So step 3 will complete within $sp + h - 1$ time slots.

Summing the above costs, we have the total cost for our algorithm:

$$T = (2sp + 2h - 1) (T_s + \frac{m}{tp} T_c) .$$

² In a tree, a terminal node is one that does not have any child.

To find the best value of p , let the derivative of T with respect to p be zero:

$$\frac{\partial T}{\partial p} = 2s(T_s + \frac{m}{tp}T_c) - (2sp + 2h - 1)\frac{m}{tp^2}T_c = 0,$$

which implies that

$$p = \sqrt{\frac{(2h-1)mT_c}{2stT_s}}.$$

However, as p is an integer, we should pick a p as follows:

$$p_{opt} = \max \left\{ 1, \left\lceil \sqrt{\frac{(2h-1)mT_c}{2stT_s}} \right\rceil \right\} \text{ or } \max \left\{ 1, \left\lfloor \sqrt{\frac{(2h-1)mT_c}{2stT_s}} \right\rfloor \right\}. \quad (3)$$

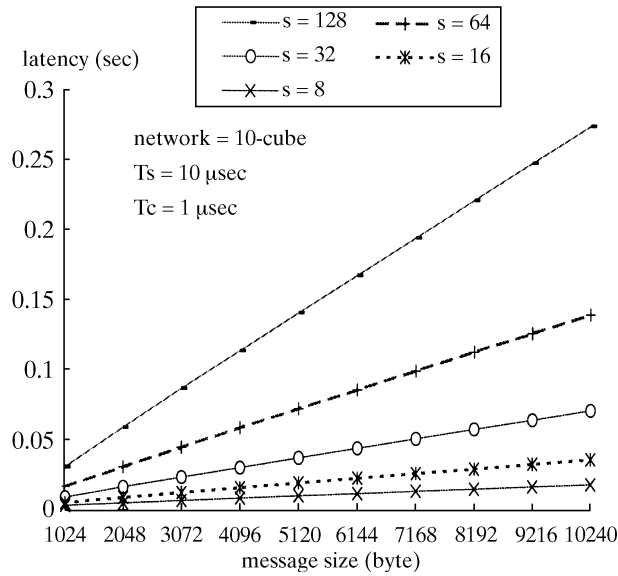
If $p_{opt} \gg 1$, the communication latency can be approximated as follows (by letting $p = p_{opt}$):

$$T_{min} = (2h-1)T_s + 2\sqrt{\frac{2sm(2h-1)T_sT_c}{t}} + \frac{2sm}{t}T_c. \quad (4)$$

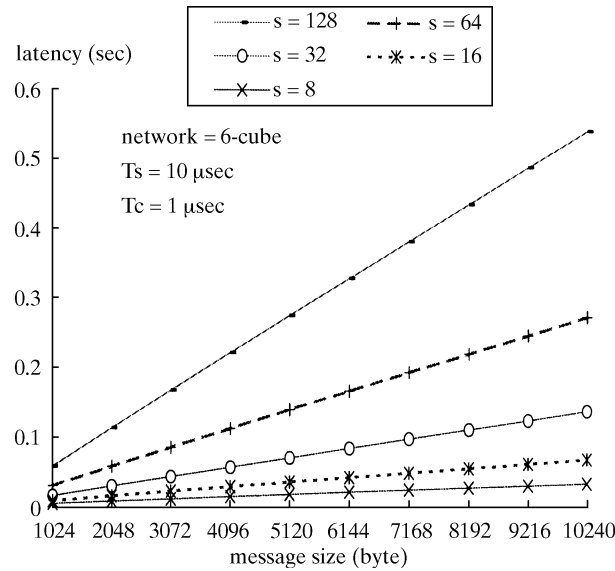
In Eq.(4), the second term is equal to two times the square root of the product of the first and third terms. Thus, T_{min} is no more than four times the maximum of the first and third terms. Comparing the first and third terms with the lower bounds in Lemma 1, we see that T_{min} is always bounded by eight times the minimal latency. In particular, when the first term of Eq.(4) dominates the third term, i.e., $(2h-1)T_s \gg \frac{2sm}{t}T_c$, we have $T_{min} \approx (2h-1)T_s$, which is at most two times the lower bounds in Lemma 1. On the contrary, when the third term of Eq.(4) dominates the first term, we have $T_{min} \approx \frac{2sm}{t}T_c$, which is also at most two times the lower bounds. Finally, we will comment on how $p_{opt} \gg 1$ can hold. Eq. (3) indicates that this will be true if mT_c is sufficiently larger than sT_s . Furthermore, if the machine-dependent factors T_s and T_c are fixed, a sufficiently large m will guarantee this condition.

In the special case of $p = 1$, the communication latency will become $(2s + 2h - 1)(T_s + \frac{m}{t}T_c) = O((s+n)(T_s + \frac{m}{t}T_c))$. Asymptotically, the factor associated with T_s could be an order of $O(\frac{s}{n})$ higher than the lower bounds, and the factor associated with T_c an order of $O(\frac{n}{s})$ higher than the lower bounds.

Fig. 1 illustrates the communication latency of our algorithm in a 10-cube and 6-star under various message sizes and numbers of sources. One nice property that can be observed from the figure is that the latency is linear to the value of m .



(a)



(b)

Fig. 1. The communication latency of our multi-node broadcasting in (a) 10-cube and (b) 6-star with the communication parameters $T_s = 10 \mu\text{sec}$ and $T_c = 1 \mu\text{sec}$.

A Comparison with a Hypercube Algorithm

Stamoulis and Tsitsiklis [12] proposed a scheme (termed the ST scheme below) to treat the same multi-node broadcasting problem for an n -cube. Although the

scheme is claimed to incur latency which is at most 2 to 6 times the optimum, the analysis in [12] ignored the startup cost, and the broadcast message was assumed to be of unit length. So we re-calculate the latency below by including these factors.

The ST scheme also uses the n edge-disjoint trees as described in [5]. However, a source node will only select one of the trees to broadcast its message. To determine which tree to use, the well-known *prefix sums* [7] are calculated. This requires $2n + 1$ phases. Assuming that only one byte of data is transmitted in calculating prefix sums, the cost will be

$$T_{prefix} = (2n + 1)(T_s + T_c).$$

Using the prefix sums, the ST scheme then evenly assigns $\lfloor \frac{s}{n} \rfloor$ or $\lceil \frac{s}{n} \rceil$ source nodes to each in-tree/out-tree. (The use of these trees is similar as ours.) Again, to apply the pipelining technique, let us partition each broadcast message (m bytes in length) into p packets. Following a similar derivation as before, this cost can be formulated as

$$T_{broadcast} = (2\lceil \frac{s}{n} \rceil p + 2n - 1)(T_s + \frac{m}{p}T_c).$$

Letting the derivative of $T_{broadcast}$ be 0, the best p occurs at

$$p_{opt} = \max \left\{ 1, \left\lceil \sqrt{\frac{(2n-1)mT_c}{2\lceil \frac{s}{n} \rceil T_s}} \right\rceil \right\} \text{ or } \max \left\{ 1, \left\lfloor \sqrt{\frac{(2n-1)mT_c}{2\lceil \frac{s}{n} \rceil T_s}} \right\rfloor \right\}. \quad (5)$$

The latency of this scheme is $T_{prefix} + T_{broadcast}$ with $p = p_{opt}$.

We observe that there are two factors affecting the performance of the ST algorithm: (i) the extra cost T_{prefix} , and (ii) whether the loads on the n edge-disjoint in-trees/out-trees are balanced or not. Factor (ii) is determined by the number of sources, s , and the dimension of the cube, n . Specifically, if s is a multiple of n , every in-tree/out-tree will have $\frac{s}{n}$ sources performing broadcasting on it. But if s is not a multiple of n , some trees may have $\lceil \frac{s}{n} \rceil$ sources. Using the number of sources as an indicator, in our algorithm, each tree always has $\frac{s}{n}$ sources. The difference can be significant when s is small. For instance, in a 10-cube, when $s = 32$, some trees will be loaded with $\lceil \frac{32}{10} \rceil = 4$ source nodes under the ST scheme while under our scheme, a tree will always have 3.2 source nodes; the ratio is $\frac{4}{3.2} = 1.25$. However, with a larger $s = 128$, the ratio becomes $\frac{13}{12.8} \approx 1.015$.

We have conducted a comparison of the ST algorithm and ours. Fig. 2 shows the case when s is a multiple of n . In general, the ST scheme works better with a larger s while ours works better with a smaller s . This can be explained by the p_{opt} values of the ST algorithm and ours. Recall Eq. (3) and Eq. (5). The p_{opt} value selected by the ST scheme is, in general, larger than ours. Under larger s ,

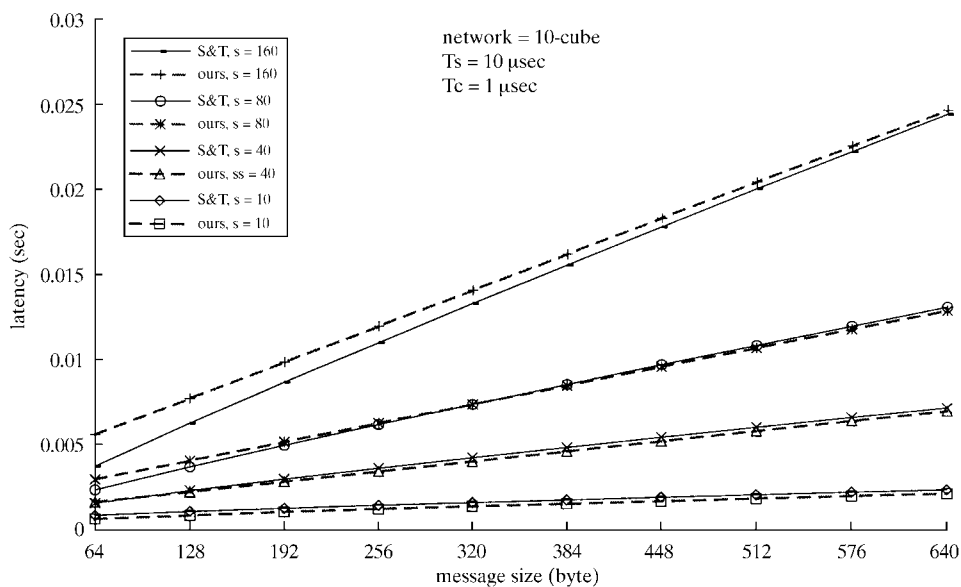


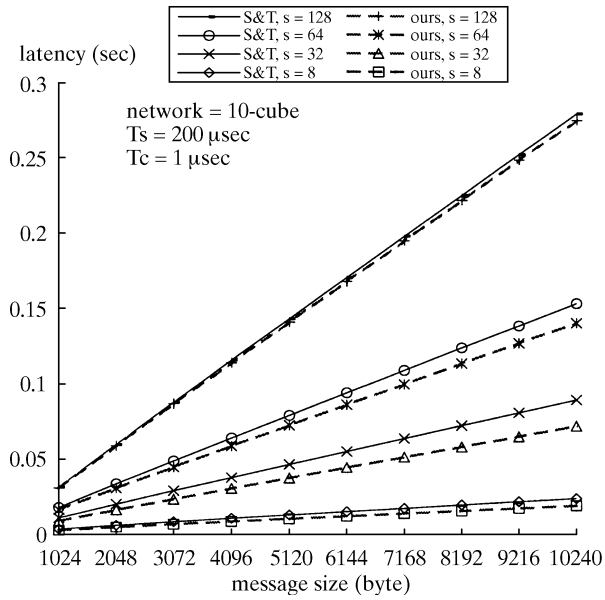
Fig. 2. Comparison of the ST algorithm and ours when s is a multiple of n .

our algorithm is more likely to choose a p_{opt} close to 1 than is the ST scheme, indicating that there is less chance that pipelined transmission will be performed in our algorithm. However, since p_{opt} will increase as m increases, this problem can be conquered as the message size increases. This scenario is reflected in the figure, which shows that our algorithm will eventually outperform the ST scheme as m becomes larger. The amount of improvement is between 5% to 10% when $s \leq 40$, and is reduced to within 3% when $s \geq 80$. This is because the cost T_{prefix} remains a constant for a fixed cube size.

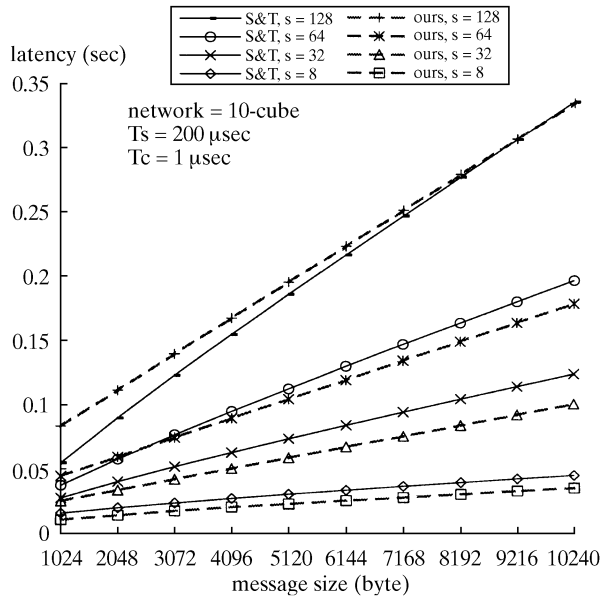
Fig. 3 shows the case when s is not a multiple of n . As can be seen, a significant gain (about 15% to 25%) can be obtained by our algorithm compared to the ST scheme when $s \leq 64$. As $s \geq 128$, the gain is reduced to within 5% since the ST scheme becomes more balanced. This conforms to our analysis. Also, comparing Fig. 3(a) and 3(b), which represent the cases of $T_s/T_c = 10$ and 200, respectively, we observe that a larger ratio of T_s/T_c also benefits the ST scheme when m is small. Again, this can be explained by the p_{opt} value, which decreases as T_s/T_c increases. Thus, we conclude that our result is more favorable than the ST scheme under smaller T_s/T_c , smaller s , and larger m ; i.e., it is more suitable for broadcasting larger messages from fewer source nodes.

5. CONCLUSIONS

We have proposed an efficient routing algorithm to solve the multi-node broadcasting problem in hypercubes and star graphs using optimal or near-optimal latency. Interestingly, the algorithm does not require knowledge in advance of the number of source nodes, nor of the locations of these nodes. When the broadcasted



(a)



(b)

Fig. 3. Comparison of the ST algorithm and ours when s is not a multiple of n under: (a) $T_s/T_c = 10$ and (b) $T_s/T_c = 200$.

messages are sufficiently large, the incurred latency is provably within 2 or 8 times the optimum. Although the algorithm has been presented for hypercubes and star graphs, it should be easy to translate the scheme for use by other network topologies if edge-disjoint spanning trees can be found (e.g., see [9] for such a possibility in a square 2-D tori).

The scheme proposed in [12] also uses n edge-disjoint spanning trees in an n -cube to solve the multi-node broadcasting problem. However, a prefix-sum operation is used to determine which source node's message should be delivered along which spanning tree. This may cause two problems. First, the traffic load on the n spanning trees may not be balanced when the number of source nodes is not a multiple of n . As shown in our simulations, our result does show a better balancing factor than does that of [12] under certain circumstances. Second, it is not easy (if not impossible) for the scheme in [12] to be extended to the more general multi-node broadcasting problem, where each source node may have a message of different length to be broadcast, as the traffic load will be more unbalanced. However, since message splitting is used, our scheme can still inject the same amount of traffic into each spanning tree and, thus, should perform better in such a situation.

Finally, we will comment on how to adjust the algorithm if the network does not provide a global physical clock. (In this case, it will be difficult for all nodes to start the algorithm at the same time.) Suppose the physical clocks in any two nodes may drift by a maximum value of δ . We can simply delay the execution of step 3 at each node by δ time. This will still keep all communications contention-free. As the value of δ is typically fairly small, the algorithm is expected to perform efficiently in this case as well.

REFERENCES

1. S. B. Akers, D. Harel and B. Krishnameurthy, "The star graph: An attractive alternative to the n -cube," in *International Conference on Parallel Processing*, 1987, pp. 393-400.
2. T. -S. Chen, Y. -C. Tseng and J. -P. Sheu, "Balanced spanning trees in complete and incomplete star graphs," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 7, 1996, pp.717-723.
3. K. Day and A. Tripathi, "A comparative study of topological properties of hypercubes and star graphs," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 1, 1994, pp.31-38.
4. P. Fragopoulou and S. G. Akl, "Edge-disjoint spanning trees on the star network with applications to fault tolerance," *IEEE Transactions on Computer*, Vol. 45, No. 2, 1996, pp.174-185.
5. S. L. Johnsson and C. T. Ho, "Optimal broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computer*, Vol. 38, No. 9, 1989, pp. 1249-68.
6. J. -S. Jwo, S. Lakshminarayanan and S. K. Dhall, "Embeddings of cycles and grids in star graphs," *Symposium on Parallel and Distributed Processing*, 1990, pp. 540-547.

7. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan Kaufmann, 1992.
8. V. E. Mendia and D. Sarkar, "Optimal broadcasting on the star graph," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 4, 1992, pp.389-396.
9. P. Michallon and D. Trystram, "Minimum depth arcs-disjoint spanning trees for broadcasting on wrap-around meshes," *International Conference on Parallel Processing*, 1995, pp. I-80-83.
10. J. Mišić and Z. Jovanovic, "Communication aspects of the star graph interconnection network," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 7, 1994, pp. 678-687.
11. K. Qiu, S. G. Akl and H. Meijer, "On some properties and algorithms for the star and pancake interconnection networks," *Journal of Parallel and Distributed Computing*, Vol. 22, pp. 16-25, 1994.
12. G. D. Stamoulis and J. N. Tsitsiklis, "An efficient algorithm for multiple simultaneous broadcasts in the hypercube," *Information Process. Lett.*, Vol. 46, 1993, pp. 219-224.
13. Y. -C. Tseng, S. -H. Chang and J. -P. Sheu, "Fault-tolerant ring embedding in a star graph with both link and node failures," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 12, 1997, pp. 1185-95.
14. Y. -C. Tseng and J. -P. Sheu, "Toward optimal broadcast in a star graph using multiple spanning trees," *IEEE Transactions on Computer*, Vol. 46, No. 5, 1997, pp. 593-599.

Yu-Chee Tseng (曾煜棋) received his B. S. and M. S. degrees in Computer Science from National Taiwan University and National Tsing Hua University in 1985 and 1987, respectively. From 1989 to 1990, he worked for the WANG Laboratory and the D-LINK Inc. as a software engineer. He obtained his Ph. D. in Computer and Information Science from the Ohio State University in January of 1994. From February of 1994 to July of 1996, he was with the Department of Computer Science, Chung-Hua University, Taiwan. Since August of 1996, he has been an Associate Professor in Department of Computer Science and Information Engineering, National Central University. Dr. Tseng served on the Program Committee of the *International Conference of Parallel and Distributed Systems* in 1996 and on the Program Committee of the *International Conference on Parallel Processing* in 1998. His research interests include parallel and distributed computing, fault-tolerant computing, parallel computer architecture, wireless network, and mobile computing.

Dr. Tseng is a member of the IEEE Computer Society and the Association for Computing Machinery.