Journal of Interconnection Networks, Vol. 1, No. 4 (2000) 303–314

DISTRIBUTED ALGORITHM FOR MULTIMESSAGE MULTICASTING

TEOFILO F. GONZALEZ
Department of Computer Science
University of California
Santa Barbara, CA, 93106
teo@cs.ucsb.edu

Received 23 May 1999
Revised 17 November 2000

We consider the multimessage multicasting over the n processor complete (or fully connected) static network when the forwarding of messages is allowed, and initially each processor only knows the messages it needs to send and their destinations. We present an efficient distributed algorithm to route the messages for every degree d problem instance with total expected communication time $O(d + \log n)$, where d is the maximum number of messages that each processor may send (or receive). Our routing algorithm consists of three phases. In the first phase the processors exchange messages to learn some basic global information. In the second phase each processor forwards its messages to transform the problem to a multimessage unicasting problem of degree d. The third phase uses a well known distributed algorithm to transmit all the resulting unicasting messages.

Keywords: Approximation algorithms, Multimessage multicasting, Forwarding, Randomized algorithms, Fully connected networks.

1. Introduction

The Multimessage Multicasting problem over the n processor static network or simply a network, MM_C, consists of sending messages in such a way that all the communications can be carried in the least total number of communication steps for every given set of messages. Specifically, there are n processors, $P = \{P_1, P_2, \ldots, P_n\}$, interconnected via a fully connected network N. Each processor is executing processes, and these processes are exchanging messages that must be routed through the links of N. We assume that processors alternate between computation and communication in a synchronous way. Our objective is to find specific times when each of these messages is to be transmitted so that all the communications can be carried in the least total number of communication steps. *Forwarding*, which means that messages may be sent through indirect paths even though a single link direct paths exist, allows communication schedules with significantly smaller total communication time. This version of the multicasting problem is referred to as the MMF_C problem, and the objective is to transmit the messages so that all the communications can be carried in the least total amount of time. In this paper we study the *distributed* version of the MMF_C, which we refer to as the $DMMF_C$ problem. In this version of the problem each processor initially knows the value of n and d, plus the messages it will be sending and their destinations. The non-distributed (or off-line) version is simpler because there is a preprocessing phase where all the information is available in one processor and this information is used to construct communication schedules that are subsequently distributed to the individual processors. In this paper we assume that each of the (original) messages to be transmitted is at least n bits long. This assumption allow us to send n-bit messages, other than the original ones, to specify forwarding information. At the end we just report the total number of messages, rather than having to report counts for the two type of messages separately. Our introduction is a condensed version of Gonzalez'[14] which includes a complete justification for the multimessage multicasting problem as well as motivations, applications, and examples.

We formally define our problem. Each processor P_i *holds* the set of messages h_i and for each of its messages $m_{i,j}$ it knows the set of processors $s_{i,j}$ that must receive the message. From this information one can compute for each processor P_i the set of messages it *needs* to receive, n_i. Note that our algorithm does not compute the n_is, but at the end each processor P_i will have all the messages it needs. We define the *degree* of a problem instance as $d = \mathbf{max}\{\mid h_i \mid, \mid n_i \mid\}$, i.e., the maximum number of messages that any processor sends or receives. Consider the following example.

Example 1: There are nine processors ($n = 9$). Processors P_1, P_2, and P_3 send messages only, and the remaining six processors receive messages only.* The messages each processor holds and needs are given in Table 1. For this example the density d is 3. Note that processors P_1, P_2, and P_3 do not need any messages, but the remaining processors each need three messages each.

Table 1. Hold and Need vectors for Example 1.

Initial messages held at each processor.

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
$\{a,b\}$	$\{c,d\}$	$\{e,f\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Messages needed at each processor.

n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
$\emptyset$	$\emptyset$	$\emptyset$	$\{a,c,e\}$	$\{a,d,f\}$	$\{b,c,e\}$	$\{b,d,f\}$	$\{c,d,e\}$	$\{c,d,f\}$

*Note that in general processors may send and receive messages.

One may visualize problem instances by directed multigraphs. Each processor P_i is represented by the vertex labeled i, and there is a directed edge (or branch) from vertex i to vertex j for each message that processor P_i needs to transmit to processor P_j. The set of directed edges or branches associated with each message are *bundled* together. The problem instance given in Example 1 is depicted in Figure 1 as a directed multigraph with additional thick lines that identify all edges or branches in each bundle.

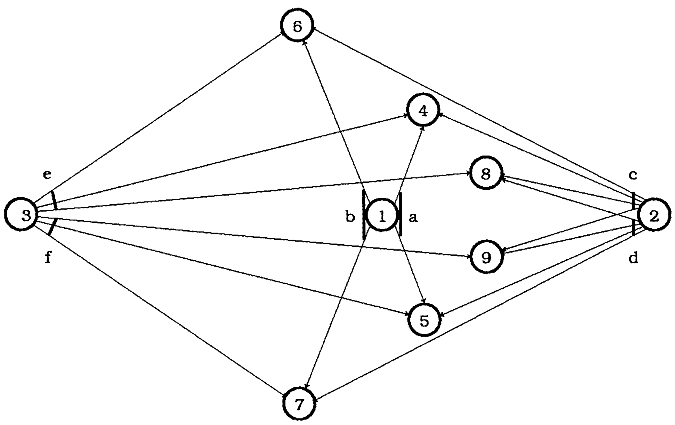

Fig. 1. Directed Multigraph Representation for Example 1. The thick line joins all the edges (branches) in the same bundle.

The communications allowed in our complete network for the distributed version of the problem must satisfy the restrictions given below. For the non-distributed versions studied in the past[10,11,14,13,15], rule 2 given below is simpler because those algorithms made sure that each processor received at most one message at a time. We should also point out that the last part of rule 2 is needed only for the third phase of our procedure, solving the resulting multimessage unicasting problem, because all the communications in the first two phases are predicatable with the information available, and thus communication conflicts can be avoided.

1.- During each time unit each processor P_i may transmit one of the messages it holds (i.e., a message in its hold set h_i at the beginning of the time unit), but such message can be multicasted to a set of processors. The message will remain in the hold set h_i.

2.- During each time unit each processor may receive at most one message. The message that processor P_i receives (if any) is added to its hold set h_i at the end of the time unit. If two or more messages are sent to a processor at a time period, then the messages are garbled and the processor does not receive any of the messages. The sending processor will know at the end of time period whether or not the message it sent reached all its destinations. Note that if the message does not reach all its destinations, then the processor will not know the processors that received the message.

The communication process ends when each processor has $n_i \subseteq h_i$, i.e., each processor holds all the messages it needs. Note that at each time unit the hold set h_i for each processor will increase by one message or remain the same depending on whether or not a new message arrives. Our communication model allows us to transmit any of the messages in one or more stages. I.e., any given message may be transmitted at different times. This added routing flexibility allow us to bound by $O(d^2)$ the total communication time[14]. In the former communication model one cannot bound the total communication time by $O(f(d))$ for any function $f(d)$[10]. The problem instance given in Example 1 requires six communication steps if one restricts each message to be transmitted only at a single time unit, but allowing messages to be transmitted at different times one can perform all communications in four steps, and if forwarding is allowed it can be further reduced to three steps[14,15]. When forwarding is allowed all the communications can be carried out in $2d$ steps[13,15].

A *communication mode* C is a set of tuples of the form (m, l, D), where l is a processor index ($1 \le l \le n$), and message $m \in h_l$ is to be multicasted from processor P_l to the set of processors with indices in D. In addition the set of tuples in a communication mode C must have the property that each processor sends at most one message at a time, and if some processor is sent two or more messages, then neither of these messages are received.

A solution to our problem instance I is a sequence of communication modes such that after performing all of these communications $n_i \subseteq h_i$ for $1 \le i \le n$, i.e., every processor holds all the messages it needs. The *total communication time* is the latest time at which there is a communication which is equal to the number of communication modes, and our problem consists of each processor sending messages in a synchronized mode so that all messages reach their destination in the least total number of communication steps. From the communication rules we know that every degree d problem instance has at least one processor that requires d time units to send, and/or receive all its messages. Therefore, d is a trivial lower bound for the total communication time.

2. Previous Work

The *basic multicasting problem* (BM_C) consists of all the degree $d = 1$ MM_C problem instances, and can be trivially solved by sending all the messages at time zero. There are no conflicts because $d = 1$, i.e., each processor sends at most one message and receives at most one message. The communication schedule has only one communication mode.

Gonzalez[14] also considered the case when each message has fixed *fan-out* k (maximum number of processors that may receive a given message). When $k = 1$ (multimessage unicasting problem MU_C), Gonzalez showed that the problem corresponds to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time, and each degree d problem instance has a communication schedule with total communication time equal to d.

It is not surprising that several authors have studied the MU_C problem as well as several interesting variations for which NP-completeness has been established, subproblems have been shown to be polynomially solvable, and approximation algorithms and heuristics have been developed. Coffman et. al.[7] studied a version of the multimessage unicasting problem when messages have different lengths, each processor has $\gamma(P_i)$ ports each of which can be used to send or receive messages, and messages are transmitted without interruption (non-preemptive mode). Whitehead[23] considered the case when messages can be sent indirectly. The preemptive version of these problems as well as other generalizations were studied by Choi and Hakimi[4,5,6], Hajek and Sasaki[18], Gopal et. al.[17]. Some of these papers considered the case when the ports are not interchangeable, i.e., it is either an input port or an output port. Rivera-Vega et. al.[20] studied, the file transferring problem, a version of the multimessage unicasting problem for the complete network when every vertex can send (receive) as many messages as the number of outgoing (incoming) links. The distributed version of the multimessage unicasting problem with forwarding, $DMUF_C$,has been studied in the context of optical-communication parallel computers[3,8,9,22]. Valiant[22] presented a distributed algorithm with $O(d + \log n)$ total expected communication cost. The algorithm is based in part on the algorithm by Anderson and Miller[3]. The communication time is optimal, within a constant factor, when $d = \Omega(\log n)$, and Gereb-Graus and Tsantilas[8] raised the question as to whether a faster algorithm for $d = O(\log n)$ exits. This question was answered in part by Goldberg et. al.[9] who show all communication can take place in $O(d + \log \log n)$ communication steps with high probability, i.e., if $d < \log n$ then the failure probability can be made as small as n^α for any constant α. Gereb-Graus and Tsantilas[8] presented distributed algorithms without forwarding with $\Theta(d + \log n \log \log n)$ expected communication steps. With the exception of a few papers[10,11,14,13,15,12,21] research has been limited to unicasting and all known results about multicasting are limited to single messages. Shen[21] has studied multimessage multicasting for hypercube connected processors. His procedures are heuristic and try to minimize the maximum number of hops, amount of traffic, and degree of message multiplexing. The MM_C problem involves multicasting of any number of messages, and its communication model allows the concurrent transmission of a large set of messages.

The MM_C problem is significantly harder than the MU_C. Gonzalez[14] showed that even when $k = 2$ the decision version of the MM_C problem is NP-complete. Gonzalez[10] developed an efficient algorithm to construct for any degree d problem instance a communication schedule with total communication time at most d^2, and presented problem instances for which this upper bound on the communication time is best possible, i.e. the upper bound is also a lower bound. The lower bound holds when there is a huge number of processors and the fan-out is also huge. Since this situation is not likely to arise in the near future, the MM_C problem with restricted fan-out has been studied[10,11].

Gonzalez[14] developed an algorithm to construct a communication schedule with total communication time $2d-1$ for the case when the fan-out is two, i.e., $k=2$. Gonzalez[14] developed an $O(q \cdot d \cdot e)$ time algorithm, where $e \leq nd$ (the input size), to construct for degree d problem instances a communication schedule with total communication time $qd+k^{\frac{1}{q}}(d-1)$, where q is the maximum number of time periods where each message can be sent and $k > q \geq 2$. Gonzalez[10,11] also developed several fast approximation algorithms with improved approximation bounds for problems instances with any arbitrary degree d, but small fan-out. The approximation bound for these methods is about $(\sqrt{k}+1)d$, where k is the fan-out.

It is simple to show that the NP-completeness reduction for the MM_C problem[14] can be easily modified to establish the NP-completeness for the MMF_C problem. All the approximation results for the MM_C problem also hold for the MMF_C problem. However, for $d > 2$ it is impossible to prove that there exists an instance of the MMF_C problem that requires d^2 communication steps. Gonzalez[13,15] presents efficient algorithms to construct for every degree d problem instance a communication schedule with total communication time at most $2d$, where d is the maximum number of messages that each processor may send (receive). We should point out that previous approximation algorithms[10,11] are faster than these ones. However, these algorithms generate communication schedules with significantly smaller total communication time. These algorithms consists of two phases. In the first phase a set of communications are scheduled to be carried out in d time periods, and when these communications are performed the resulting problem is a degree d multimessage unicasting problem. The second phase generates a communication schedule for this problem by reducing it to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time. The solution is the concatenation of the communication schedules for each of these two phases. For $2 \leq l \leq d$, Gonzalez[15] defined the l-MMF_C as the MMF_C in which each processor has at most ld edges emanating from it and presented an algorithm to generate a communication schedule with total communication time at most $\lfloor (2-\frac{1}{l})d \rfloor + 1$ for the l-MMF_C problem.

In this paper we study the $DMMF_C$ problem. In this version of the problem each processor initially knows the value of n and d, plus the messages it will be sending and their destinations. The algorithm is a combination of algorithms including the classic parallel prefix algorithm[19], the message forwarding phase of Gonzalez' algorithm[15] for multimessage multicasting with complete information, and Valiant's[22] distributed algorithm for the multimessage unicasting problem. The result is a distributed algorithm to route all messages with $O(d+\log n)$ expected communication steps. One can also use Goldberg et. al.[9] algorithm instead of Valiant's[22] algorithm. We use the latter because it is simpler.

3. Approximation Algorithm for the $DMMF_C$ Problem

Given an instance of the $DMMF_C$ problem we present our strategy based on the classic parallel prefix algorithm[19], the message forwarding phase of Gonzalez' algorithm[15] for multimessage multicasting with complete information, and Valiant's[22] distributed algorithm for the multimessage unicasting problem. The result is a distributed algorithm to route all messages in $O(d + \log n)$ expected communication steps. Remember that we have assumed that the (initial) messages have length at least n bits long, and that every processor knows the value of d and n.

Our strategy is to use the classic parallel prefix algorithm[19] to compute and exchange information, and then use this information to run a distributed version of Gonzalez' algorithm[15] with partial global information. By forwarding all the messages, Gonzalez' algorithm[15] transforms the problem to a multimessage unicasting problem. All of the resulting communications can be performed by Valiant's[22] distributed algorithm.

Before we proceed it is important to understand the message forwarding phase of Gonzalez' algorithm[15] that reduces the problem to a multimessage unicasting problem. We explain how this phase works by applying it to the problem instance given in Figure 2. The problem instance consists of 12 processors, 11 messages, and has degree $d = 2$.

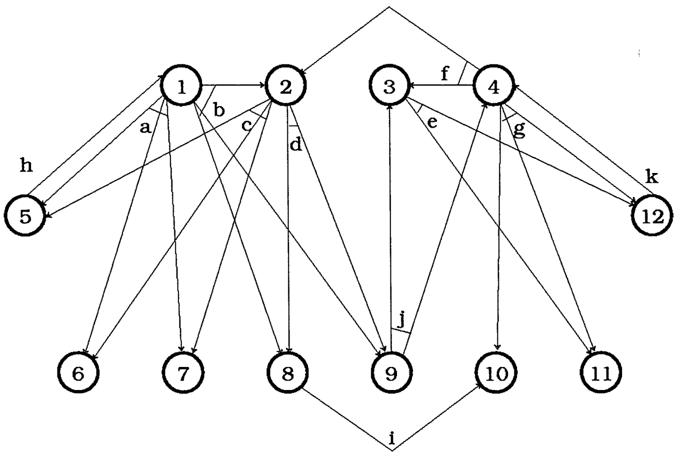

Fig. 2. MM_C problem instance (I, G).

In Figure 3 we show all the processors with a list of labels assigned to the bundles and edges that are defined as follows. The top set of numbers is the bundle number which is defined by labeling the bundles emanating out of processor P_1, then the one emanating out of P_2, and so forth. The next label is the message for the bundle and the third one is the bundle number modulo (d) plus 1. This third number is the time at which the message associated with the bundle will be forwarded. From the way these labels are generated, we know that no two bundles emanating out of a processor will forward a message at the same time. The edges are labeled beginning with the ones emanating out of the first bundle, then the second one, and so forth. These labels are shown in the fourth line. The last set of numbers is the ceil of the edge number divided by d. This last row indicates the processor index where the message will be forwarded. It is simple to see that each processor will receive at most d messages and all these messages will be received at different times. In what follows we explain in detail the application of this procedure to the problem instance in Figure 3.

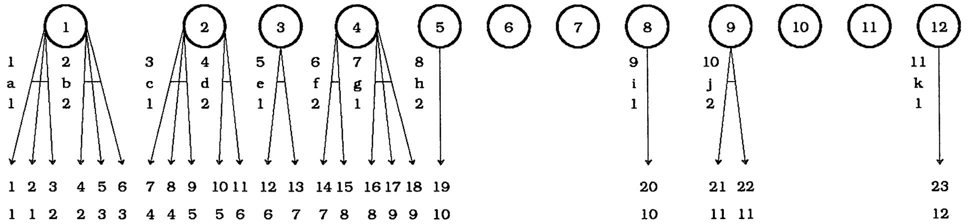

Fig. 3. Labeling performed by procedure FORWARD.

At time 1 message a is multicasted from processor P_1 to processors P_1 and P_2. Obviously one does not actually need to send the message to processor P_1, since P_1 holds that message. Our algorithm could be modified to detect cases like this one, but in general the total communication time will not decrease. In what follows we will only make minor comments when this type of situations arises. At time 1 message c is multicasted from processor P_2 to processors P_4 and P_5; message e is multicasted from processor P_3 to processors P_6 and P_7; message g is multicasted from processor P_4 to processors P_8 and P_9; message i is unicasted from processor P_8 to processor P_{10}; and message k is unicasted from processor P_{12} to processor P_{12} (superfluous operation). All of these communications are represented by the forest labeled $T1$ in Figure 4. The specific communication operations for time 2 are given in the forest labeled $T2$ in Figure 4.

The resulting unicasting problem $(\hat{I}, \hat{G})$ of degree d is given in Figure 5 (all objects). Since the leftmost two edges in the bundle B_1 were forwarded to processor P_1 (superfluous operation), then message a is to be sent from processor P_1 to processor P_5 and P_6 in $(\hat{I}, \hat{G})$; the rightmost edge in bundle B_1 was forwarded to processor P_2, therefore message a needs to be sent to processor P_7 from P_2; the leftmost edge in bundle B_2 was forwarded to processor P_2, therefore message b needs to be sent to processor P_8 from P_2; the rightmost two edges in bundle B_2 were forwarded to processor P_3, therefore message b needs to be sent to processors P_9 and P_2 from P_3; the leftmost two edges in bundle B_3 were forwarded to processor P_4, therefore message c needs to be sent to processors P_5 and P_6 from P_4; the rightmost edge in bundle B_3 was forwarded to processor P_5, therefore message c needs to be sent to

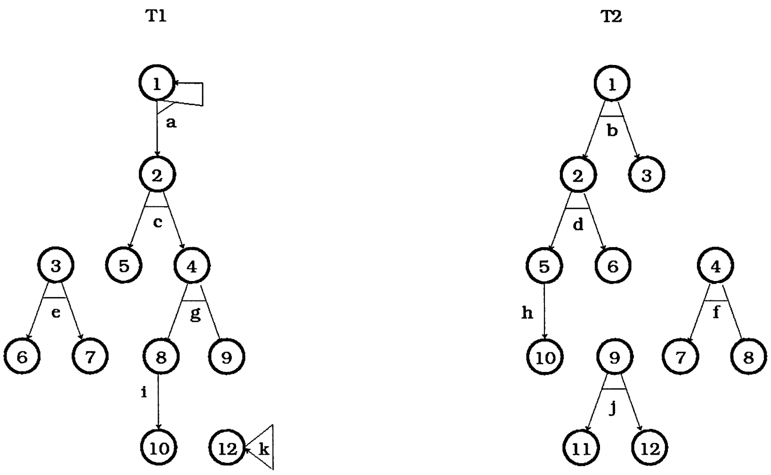

Fig. 4. Communications at time one ($T1$) and time two ($T2$).

processor P_7 form P_5; the two edges in bundle B_4 were forwarded to processors P_5 and P_6, therefore message d needs to be sent to processors P_8 and P_9 from P_6; and so on. The resulting unicasting problem ($\hat{I}$, $\hat{G}$) of degree d is given in Figure 5.

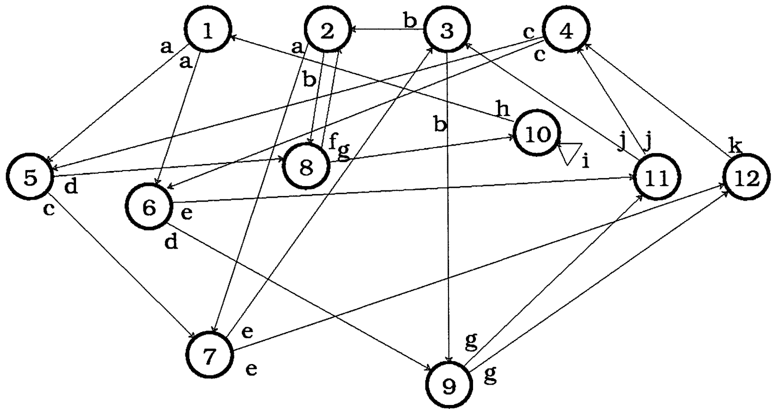

Fig. 5. MU_C problem instance $(\hat{I}, \hat{G})$ constructed from (I, G) in Figure 2.

Remember that Gonzalez' algorithm[15] is non-distributed and all the information is known globally. However, the algorithm in this paper is distributed and the only information every processor knows initially are the messages it needs to send and their destinations, and the values for n and d. We now discuss our algorithm.

1. **Compute and Broadcast Basic Information.** Each processor P_i needs to know the total number of messages that processors $P_1, P_2, \ldots, P_{i-1}$ need to send as well as the total number of destinations for all of these messages. I.e., the total number of bundles and the total number of edges emanating out of all of these processors. This information is needed to label the bundles and edges emanating out of P_i, and it can be easily computed via the classic parallel prefix[19] in $O(\log n)$ communication steps.

2. **Transform to the Multimessage Unicasting Problem via Gonzalez' Algorithm[15].**

   We transform Gonzalez' algorithm[15] into a distributed one.

Procedure FORWARD for P_j

/* the value of n and d are known in every processor */
/* The following information computed in Step 1 is available in P_j
  n_b: total number of bundles emanating from processors $P_1, P_2, \ldots, P_{j-1}$.
  n_e: total number of edges emanating from processors $P_1, P_2, \ldots, P_{j-1}$.
*/
Label B_{n_b+i} the i^{th} bundle visited while traversing the bundles emanating from P_j;
Define $t(n_b+i)$ as $(n_b+i-1) \bmod (d)+1$;
/* The message associated with bundle B_{n_b+i} will be forwarded at time $t(n_b+i)$. */
Label e_{n_e+i} the i^{th} edge visited while traversing the bundles emanating from P_j in the order $B_{n_b+i}, B_{n_b+i+1}, \ldots$;
Define the function $g(n_e+i)$ as $\lceil \frac{n_e+i}{d} \rceil$;
/* Edge e_{n_e+i} will be forwarded to processor $P_{g(n_e+i)}$ */
for every bundle B_{n_b+i} emanating from P_j **do**
  Let $S_{n_b+i} = \{g(l) | e_l \in B_{n_b+i}\}$;
endfor
for $t = 1, 2, \ldots, d$ **do**
  if there is a bundle emanating out of P_j with $t(n_b+i) == t$ **then**
    At time t processor P_j multicasts message B_{n_b+i} to the set of processors S_{n_b+i} (if $|S_{n_b+i}| = 1$, the operation is unicasting). Appended to this message one sends a bit vector of size n indicating the processor indices of the processors that will eventually receive this message, as well as the first processor that will receive this forwarded message and the number of edges that such processor must forward.
    /* This info is used by the forwarding processors to compute the destinations of the messages being forwarded. */
endfor
end of Procedure

3. **Solving the resulting Multimessage Unicasting Problem Instance.**
   At this point each processor just runs Valiant's algorithm[22] and all the messages are delivered to their destinations in $O(d + \log n)$ expected communication steps.

Lemma 3.1. The pair $(\hat{I}, \hat{G})$ is a problem instance of the MU_C problem and once all its messages are transmitted will solve the original multimessage multicasting problem (I, G).

Proof. The proof of the lemma is based on the observations that the $t()$ and $g()$ labels computed by our procedure are identical to the one that Gonzalez' algorithm[15] would have computed for this problem instance. This implies that the messages will be forwarded exactly as in Gonzalez' algorithm[15]. Therefore, solving the resulting multimessage unicasting problem solves the original problem. □

Theorem 3.1. Our algorithm performs all the multicasting for every instance of the $DMMF_C$ problem with $O(d + \log n)$ expected communication steps.

Proof. The proof of the theorem is based in the previous lemma, and the correctness proof of the subprocedures used by our algorithm. The total number of communication steps in phase 1 is $O(\log n)$, and in phase 2 is $O(d)$. The number of expected communication steps for phase 3 is $O(d + \log n)$. □

4. Discussion

The most important open problem is to develop an efficient distributed algorithms with similar performance guarantees but for the case when the processors are connected via a dynamic network where the communication elements can replicate data. The non-distributed version of this problem has already been solved by Gonzalez[15]. The main difficulty in extending that work to the distributed case is the construction of the routing tables with only local information.

References

1. G. S. Almasi, and A. Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings, New York, 1994.
2. S. G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall Inc. 1989.
3. R. J. Anderson, and G. L. Miller, Optical Communications for Pointer Based Algorithms, Technical Report CRI 88 – 14, CS Department, University of Southern California, Los Angeles, 1988.
4. H.-A. Choi, and S. L. Hakimi, "Scheduling File Transfers for Trees and Odd Cycles," *SIAM Journal on Computing*, Vol. 16, No. 1, (1987), pp. 162 – 168.

5. H.-A. Choi, and S. L. Hakimi, "Data Transfers in Networks with Transceivers," *Networks*, Vol. 17, (1987), pp. 393 – 421.
6. H.-A. Choi, and S. L. Hakimi, "Data Transfers in Networks," *Algorithmica*, Vol. 3, (1988), pp. 223 – 245.
7. E. G. Coffman, Jr, M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling File Transfers in Distributed Networks," *SIAM J. on Computing*, Vol. 14, No. 3, (1985), pp. 744 – 780.
8. M. Gereb-Graus and T. Tsantilas, "Efficient Optical Communication in Parallel Computers," Proceedings 4th ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1992, pp. 41 – 48.
9. L. A. Goldberg, M. Jerrum, T. Leighton, and S. Rao., "Doubly Logarithmic Communication Algorithms for Optical-Communication Parallel Computers," *SIAM J. Computing*, Vol. 26, No. 4, (1997), pp. 1100 – 1119.
10. T. F. Gonzalez, "Multi-Message Multicasting," Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems (Irregular'96), Lecture Notes in Computer Science (1117), Springer, (1996), pp. 217-228.
11. T. F. Gonzalez, "Improved Multimessage Multicasting Approximation Algorithms," Proceedings of the Ninth International Conference on Parallel and Distributed Computing Systems PDCS'96, (1996), pp. 456 – 461, July 1996.
12. T. F. Gonzalez, "Improved Approximation Algorithms for Multimessage Multicasting," UCSB Department of Computer Science, Technical Report TRCS-96-16, July 1996.
13. T. F. Gonzalez, "Algorithms for Multimessage Multicasting With Forwarding," Proceedings of the Tenth International Conference on Parallel and Distributed Computing Systems PDCS'97, (1997), 372 – 377.
14. T. F. Gonzalez, "Complexity and Approximations for Multimessage Multicasting," *Journal of Parallel and Distributed Computing,* Vol. 55, No. 2, (1998), 215 – 235.
15. T. F. Gonzalez, "Simple Multimessage Multicasting Approximation Algorithms With Forwarding," *Algorithmica*, (to appear).
16. T. F. Gonzalez, and S. Sahni, "Open Shop Scheduling to Minimize Finish Time," *JACM*, Vol. 23, No. 4, (1976), pp. 665 – 679.
17. I. S. Gopal, G. Bongiovanni, M. A. Bonuccelli, D. T. Tang, and C. K. Wong, "An Optimal Switching Algorithm for Multibean Satellite Systems with Variable Bandwidth Beams," *IEEE Transactions on Communications*, COM-30, No. 11 (1982) pp. 2475 – 2481.
18. B. Hajek, and G. Sasaki, "Link Scheduling in Polynomial Time," *IEEE Transactions on Information Theory*, Vol. 34, No. 5, (1988), pp. 910 – 917.
19. R. E. Ladner, and M. J. Fischer, "Parallel Prefix Computation," *Journal of th ACM,* Vol. 27, 1980, pp. 831 – 838.
20. P. I. Rivera-Vega, R, Varadarajan, and S. B. Navathe, "Scheduling File Transfers in Fully Connected Networks," *Networks*, Vol. 22, (1992), pp. 563 – 588.
21. H. Shen, "Efficient Multiple Multicasting in Hypercubes," *Journal of Systems Architecture,* Vol. 43, No. 9, Aug. 1997, pp. 655 – 662.
22. L. G Valiant, "General Purpose Parallel Architectures," Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, New York, 1990, Chapter 18 (see p 967).
23. J. Whitehead, "The Complexity of File Transfer Scheduling with Forwarding," *SIAM Journal on Computing* Vol. 19, No 2, (1990), pp. 222 – 245.