Continuous Delivery Message Dissemination Problems under the Multicasting Communication Mode

Teofilo F. Gonzalez, *Member*, *IEEE*

Abstract—We consider the Continuous Delivery Message Dissemination (*CDMD*) problem over the n-processor single-port complete (all links are present and are bidirectional) static network with the multicasting communication primitive. This problem has been shown to be NP-complete, even when all messages have the same length. For the *CDMD* problem, we present an efficient approximation algorithm to construct a message-routing schedule with a total communication time of at most $3.5d$, where d is the total length of the messages that each processor needs to send or receive. The algorithm takes $O(qn)$ time, where n is the number of processors and q is the total number of messages that the processors receive.

Index Terms—Network communications, multicasting, message routing, approximation algorithms, message forwarding.

1 Introduction

Parallel and distributed systems were introduced to accelerate the execution of programs by a factor proportional to the number of processing elements. To accomplish this goal, a program must be partitioned into tasks, and the communications that must take place between these tasks must be identified to ensure correct execution of the program. To achieve high performance, one must assign each task to a processing unit (statically or dynamically) and develop communication programs to efficiently perform all the intertask communications. Efficiency depends on the algorithms used to route messages to their destinations, which is a function of the underlying communication network, its primitive operations, and the communication model. In general terms, a message dissemination problem consists of a network with a communication model, a set of communication primitives, and a set of messages that need to be exchanged. The objective is to find a schedule to transmit all the messages in the least total number of communication rounds. In the Continuous Delivery Message Dissemination (*CDMD*) problem, each message has a length and is partitioned into packets; *however, the packets of every message must arrive at its destination in its "original" order, and all packets for each message must arrive during consecutive time units.* One may think of the messages as "video clips" to be viewed without delay on arrival (because of limited storage capability) or data that needs to be processed online in the order it is generated. Generating an optimal communication schedule, that is, one with the least total communication rounds, for the *CDMD* problem, even when all the messages have the same length, over a wide range of communication networks is an NP-hard problem. To cope with intractability, efficient message dissemination approximation algorithms for classes of networks under different communication assumptions have been developed. These algorithms may be used for a different version of the *CDMD* problem where the packets may arrive to their destinations at any time and in any order.

In this paper, we consider the *CDMD* problem. We present an efficient approximation algorithm to construct a message-routing schedule with a total communication time (TCT) of at most $3.5d$, where d is the total length of the messages that each processor may send (or receive). The algorithm takes $O(nq)$ time, where n is the number of processors and q is the total number of messages that the processors receive.

Before we formally define the *CDMD* problem, we define the communication network, the communication model, and the communication primitives under the version of the *CDMD* problem we consider in this paper. The communication network is the n-processor complete static (all links are present and are bidirectional) network N. The communication model is the *single-port* model where every processor sends at most one message and receives at most one message during each communication round. The communication primitive is called *multicasting*, which means that the message a processor sends at time t may be concurrently sent to a set of processors. All the messages take one communication round to reach their destination, regardless of the source or destination processor.

Let us formally define the *CDMD* problem. The problem consists of constructing offline a communication schedule with the least TCT for transmitting any given set of (multidestination) messages through an n-processor single-port complete static network N with the multicasting communication primitive. Specifically, there are n-processors, $P = \{P_1, P_2, \ldots, P_n\}$, interconnected via network N capable of multicasting messages through a

• *The author is with the Department of Computer Science, University of California, Santa Barbara, CA 93106-5110. E-mail: teo@cs.ucsb.edu.*

Manuscript received 8 June 2007; revised 21 Sept. 2007; accepted 10 Oct. 2007; published online 1 Nov. 2007.
Recommended for acceptance by Y. Pan.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-06-0186.
Digital Object Identifier no. 10.1109/TPDS.2007.70801.

TABLE 1
Hold and Need Vectors for Example 1.1

h_1	h_2	h_3	h_4	h_5	h_6	h_7
$\{A, B, C\}$	$\{D, E, F\}$	$\{G, H, I\}$	$\{J, K, L\}$	$\{M, N, O\}$	$\{P, Q\}$	$\{R, S\}$
n_1	n_2	n_3	n_4	n_5	n_6	n_7
$\{E, O, G\}$	$\{M, R, A\}$	$\{P, S, C\}$	$\{N, R, Q, C, B\}$	$\{R, J, A, F\}$	$\{C, K, S, I\}$	$\{R, S\}$

single port. Each processor P_i initially *holds* the set of messages h_i and *needs* to receive the set of messages n_i. Message j has length l_j (positive integer), which is the number of packets. *The packets of every message must be received in order and during consecutive time units.* The objective is to construct a communication schedule with the least *TCT*, that is, the least total number of communication rounds. At each communication round, a processor may multicast to a set of processors one of the messages it holds (that is, a message in its hold set h_i). The message will also remain in the hold set h_i. During each communication round, each processor may receive at most one message. The message that processor P_i receives (if any) will be available in its hold set h_i for the next as well as all the subsequent communication rounds. The communication process ends when each processor has $n_i \subseteq h_i$, that is, each processor holds all the messages it needs.

We assume that $\bigcup h_i = \bigcup n_i$, and that each message is initially in exactly one set h_i. We define the total length of the messages that any processor sends, or *max hold*, as $s = \mathbf{max}\{\sum_{j \in h_i} l_j\}$. It is important to note that the message length of each multicast (message with multiple destinations) counts only once in the summation for the length of the messages emanating out of each processor, which is used in the computation of s. The reason is that multicasts can be concurrently sent. The total length of the messages that any processor must receive, or *max need*, as $r = \mathbf{max}\{\sum_{j \in n_i} l_j\}$. We define the *degree* of a problem instance as $d = \mathbf{max}\{s, r\}$, that is, the maximum total length of the messages any processor sends or must receive. Clearly, for any problem instance, d is a lower bound for the TCT of any communication schedule. We use m to denote the total number of different messages, and q is the total number of messages that all processors must receive, that is, $\sum_i |n_i|$. Consider the following example. As we develop our algorithm, we will apply it to this example and generate a final schedule for it.

Example 1.1. There are seven processors $(n = 7)$, the total number of messages $m = 19$ (labeled A through S), and the total number of messages that processors must receive is $q = 24$. The messages that each processor holds and needs are given in Table 1. The length (in packets) of the messages is given in Table 2. For this example, $d = s = r = 60$.

Usually, one visualizes these types of problems by directed multigraphs with bundled edges. Each processor P_i is represented by the vertex labeled i, and each message is represented by a set of directed edges (or branches) from the sending processor to each of the receiving processors. The set of directed edges or branches associated with each message are *bundled* together. The problem instance given in Example 1.1 is depicted in Fig. 1a as a directed multigraph with additional thick lines that identify all edges or branches in each bundle. The total number of messages that processors must receive, q, is the total number of edges.

Our communication model allows us to transmit any of the messages to its destinations at the same time or at different times (any message may be transmitted to a subset of destinations starting at time t_1, to another subset of destinations starting at time t_2, and so on). This added routing flexibility normally reduces the TCT, and in many cases, it is a considerable reduction [1]. Our communication model allows for *forwarding* of messages, that is, messages may be sent to their destinations through intermediate processors that do not actually need to receive these messages. As we point out in Section 2, forwarding messages allows for schedules with smaller, and, in many cases, significantly smaller, TCTs, even when the network is fully connected and the communication load is balanced.

TABLE 2
Length of Messages in Example 1.1

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	2	23	13	31	2	14	31	3	3	3	3	5	12	15	6	3	20	31

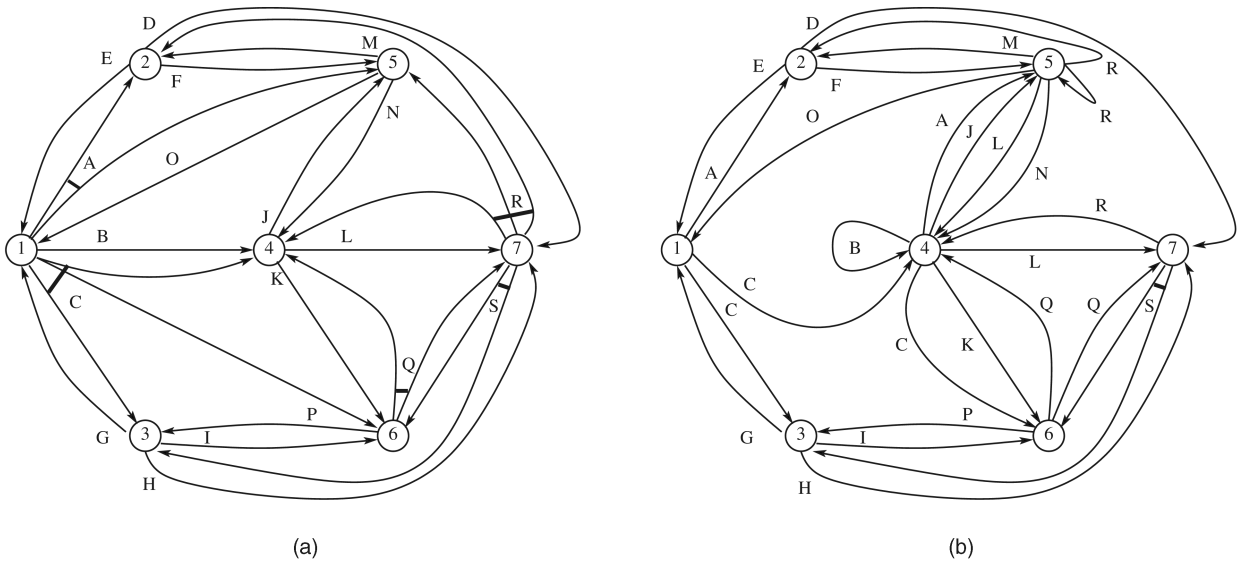

Fig. 1. (a) Multigraph for Example 1.1. (b) Resulting multigraph for Example 1.1 after forwarding messages using transformation LH given in Section 3.1. Self edges may be eliminated.

Algorithms for the completely connected architecture have wide applicability in the sense that the schedules generated for this architecture can easily be translated to communication schedules for every pr-network—a large family of communication networks [1], [2]. The class includes sets of processors connected through Benes networks (for example, the Meiko CS-2 and IBM GF-11). There is some penalty that one has to pay for the translation process that is doubling the communication rounds [1]. However, this penalty is not always incurred [2].

Most of the previous work is for a restricted version of the multicasting communication primitive, which is called the *telephone communication mode*. In this restricted communication mode, each processor can send at most one message to at most one destination at a time. As it has been established earlier, the TCT for optimal schedules under the telephone communication mode is larger than the one over the multicasting communication mode; in many cases, it is significantly larger [1]. The multicasting communication mode is available in multiprocessor computer systems. Because of the significant difference between the multicasting and telephone communication primitives, one cannot compare the algorithms for these communication modes. For example, consider the problem when one message has to be sent from one processor to all the other processors in a complete network. Under the multicasting communication primitive (the model in this paper), the communication can be carried in one step. Although in the telephone communication mode, it requires about $\log n$ steps. This fundamental difference in the quality of the solutions generated makes it impossible to make comparisons. For example, a communication schedule with a near-optimal TCT under the telephone communication mode has communication time that is much larger than the one of a communication schedule with a TCT twice that of an optimal communication schedule under the multicasting communication mode.

The *multiport communication model* is more general than the single-port one used in this paper. In this communication model, every processor may send and receive multiple messages during each communication round. This more general communication model allows for schedules with smaller a TCT; however, it requires expensive communication hardware. The additional expense is proportional to the maximum number of concurrent messages that each processor may send or receive.

2 Applications and Previous Results

A restricted version of the *CDMD* problem, where all messages have the same length, is called the *multimessage multicasting* MM_C problem. A restricted version of the MM_C problem is the *all-to-all* communication problem (also known as the *gossiping* problem) when each processor is restricted to send one message to all the other processors [3], [4]. The *all-to-many* and *many-to-many* [5] communication problems are restrictions of the *all-to-all* communication problem when message destinations and/or sources are limited to subsets of processors.

The MM_C problem, where all messages have exactly one destination, is called the *multimessage unicasting* MU_C problem. The MU_C is the *all-to-all personalized* [6] or the *all-to-many personalized* [7] communication problem when combining all the individual messages to be sent from processor i to processor j into a single one whose length is the sum of all the individual messages. Note that for the MU_C, *all-to-all personalized* and *all-to-many personalized* communication problems, the multicasting communication mode does not provide any advantages over the telephone communication mode. The reason is that every message has a single destination.

The MU_C problem and its variants have been extensively studied. The basic results include heuristics, approximation algorithms, polynomial time algorithms for restricted versions of the problem, and NP-completeness results.

Coffman et al. [8] present approximation algorithms for a generalization of the MU_C problem where there are $\gamma(P_i)$ sending or receiving ports per processor (multiport communication model). In this version of the problem, the packets from each message must arrive in order to their destinations during consecutive time units, as in our *CDMD* problem. However, our *CDMD* problem is more general in the sense that the communication pattern allows for multidestination messages. In general, for MU_C problem instances, one may construct schedules with a significantly smaller TCT when message forwarding is allowed, as initially reported by Whitehead [9].

When one relaxes the requirement that the packets from each message must arrive in order to their destinations during consecutive time units, schedules with a smaller TCT may be constructed. In this case, we say that *message splitting* or *preemptive scheduling* is allowed; that is, messages may be transmitted with interruptions. Several preemptive scheduling versions of a generalized MU_C problem have been considered by Choi and Hakimi [10], Hajek and Sasaki [11], and Gopal et al. [12] (for satellite communications). The n-port MU_C problem over complete networks, for transferring files, was studied by Rivera-Vega et al. [13]. An extensive discussion of all of these results appears in [4]. Another variation of the MU_C problem, called *the message exchange problem*, has been studied by Goldman et al. [4]. The communication model in this version of the problem is asynchronous, and it closely corresponds to actual distributed memory machines. A restricted version of the MU_C problem, called *data migration*, was studied by Hall et al. [14].

The main difference between the above mentioned research and the one that we discuss in this paper is that we concentrate on the multicasting communication mode, rather than on the telephone communication mode. Furthermore, we restrict to nonsplitting of messages (nonpreemptive schedules). Because of this difference in the communication model, one cannot directly compare the quality of the solutions generated by known algorithms for all of these problems. Almost all of the work for all-to-all personalized communication (gossiping) problem is for the telephone communication primitive (see [4] and [15] for an extensive discussion). The only exceptions are the results reported in [15]. However, the interesting point is the applicability of the algorithms to all sorts of networks, not just the complete network. Most of the previous research based on the multicasting communication mode has concentrated on routing a single message, which is a version of the Steiner tree problem. There is an extensive literature on the Steiner tree problem, but it is irrelevant to the work presented in this paper. Multimessage multicasting problem under the multicasting communication mode but involving equal-length messages is reported in [1], [2], [16], [17], [18], [19], and [20]. The initial research by Shen [19] on this type of problems was for n-cube-processor systems. The objective function was very general and attempted to minimize the maximum number of hops, amount of traffic, and degree of message multiplexing, so only heuristics could be developed.

Different strategies for solving multimessage multicasting problems over all-optical networks were surveyed by Thaker and Rouskas [20]. The multimessage multicasting problem based on the telephone communication mode has been studied under the name of *data migration with cloning* by Khuller et al. [21]. This problem arises when messages are movies being relocated to different servers in order to better satisfy users' requests [22]. A generalization of this message dissemination problem allowing for limited broadcasting or multicasting was considered by Khuller et al. [22]. These problems are used to model networks of workstations and grid computing. These data migration problems have also been studied under the objective of minimizing the weighted sum of the message arrival times by Gandhi et al. [23]. This version of the message dissemination problem is also based on the telephone communication mode.

The multimessage multicasting problem arises naturally when solving large scientific problems via iterative methods in a parallel or distributed computing environment (for example, solving large sparse system of linear equations using stationary iterative methods). Another application arises when executing the most dynamic programming procedures in a parallel or distributed computing environment. Dynamic programming is widely used in bioinformatics, operations research, computer science, etc. In information systems, these problems arise naturally when multicasting information over a b-channel ad hoc wireless communication network. Other applications include sorting, sparse matrix multiplication, discrete Fourier transform, molecular dynamics [24], etc. Message-routing problems under the multicasting communication primitive arise in sensor networks, which are simply static or slow-changing ad hoc wireless networks. This type of networks has received considerable attention because of applications in battlefields, emergency disaster relief, etc. Ad hoc wireless networks are suited for many different scenarios, including situations where it is not economically viable to provide Internet or Intranet wired communication. Other applications in high-performance communication systems include voice and video conferencing, operations on massive distributed data, scientific applications and visualization, high-performance supercomputing, medical imaging, etc. The need to deliver multidestination (multicasting) messages is expected to rapidly increase in the near future. The nonpreemptive scheduling mode in the *CDMD* problem has additional applications when the cost of preemptions is large, or when continuous processing is required, for example, in online computation or when the messages are video clips that need to be viewed as they arrive because the device receiving the video clips has limited storage. A rough version of the algorithms presented in this paper appears in [25].

Gonzalez [1] shows that even restricted versions of the multimessage multicasting (MM_C) problem are NP-complete, but schedules with a TCT of at most d^2 can be constructed in $O(q)$ time. This bound is best possible in the sense that for all $d \geq 1$, there are problem instances that require d^2 communication time [1]. When *forwarding* is allowed, the multimessage multicasting (MM_C) problem remains NP-complete, but schedules with a TCT of at most $2d$ can be constructed in $O(q(min\{q, n^2\} + n \log n)$ time [2].

3 Approximation Algorithm for the $CDMD$ Problem

We present an efficient algorithm that, for every instance of the *CDMD* problem, constructs a schedule with a TCT of at most $3.5d$. We begin by defining restricted versions of the *CDMD* problem, which will be instrumental for generating suboptimal solutions, and outline our approach to generate such solutions.

The *single-destination CDMD* ($CDMD_{SD}$) problem is the *CDMD* such that every processor sends each of its messages to only one destination. The $CDMD_{MSD}$ problem is a slight variation of the $CDMD_{SD}$ problem, where every processor has at most one multidestination message (one multicast) and the length of the message is more than $d/2$. Because of this last property, all the multidestination messages (originating at all the processors) have different destinations. When we refer to the parameters of an instance of the $CDMD_{SD}$ or $CDMD_{MSD}$ problems, we use s', r', d', n', m', and q' to refer to the corresponding parameters of the *CDMD* problem. As in the case of the *CDMD* problem, the length of the multicast is included once in the summation of the length of the messages emanating from each processor. This is used in the computation of s'. The reason being is that the multicasts may be sent concurrently.

Let us begin by trying to apply the approach used to construct schedules with a TCT of at most $2d$ for the MM_c when forwarding is allowed. The idea is to forward messages to other processors so that we are left with an instance of the $CDMD_{SD}$ problem in which every processor must send messages with a total length of at most d. This may be achievable by a forwarding schedule with a TCT of at most d. A schedule with a TCT of at most $2d$[1] can be constructed for any instance of the $CDMD_{SD}$ problem in polynomial time by the algorithm given in Section 3.2. This results in a schedule for the whole problem with a TCT of at most $3d$. However, this approach is not always possible, and just determining whether or not this approach is feasible for a problem instance is an NP-complete problem in the strong sense because the construction of the $CDMD_{SD}$ problem instance involves solving a bin-packing problem instance.

We need to modify the approach that we just discussed. In Section 3.1, we show that given any instance I of the *CDMD* problem of degree d, it is always possible to construct an instance $f(I)$ of the $CDMD_{MSD}$ problem with $s' \leq 1.5d$ and $r' \leq d$. The algorithm takes $O(nq)$ time and consists of a set of message multicasting forwarding operations. We show that this forwarding operation can be carried out by a schedule S with a TCT of d. Then, in Section 3.2, we show how to construct a schedule T with a TCT of at most $s' + r'$ for every instance of the $CDMD_{MSD}$ problem. Schedule S, followed by schedule T for the instance $f(I)$ of the $CDMD_{MSD}$, is a schedule for instance I of the *CDMD* problem with a TCT of at most $3.5d$ (schedule S has a TCT of at most d and schedule T has a TCT of at most $s' + r' \leq 2.5d$). We state this result in the following theorem whose proof follows from Theorems 3.2 and 3.3.

1. Note that for the MM_C problem with forwarding, one can construct a schedule with a TCT of at most d because all the messages are of equal length.

TABLE 3
The md-Pairs at Each Processor for Example 1.1

Processor	md-pairs
1	(A,2), (A,5), (B,4), (C,3), (C,6)
2	(D,7), (E,1), (F,5)
3	(G,1), (H,7), (I,6)
4	(J,5), (L,7), (K,6)
5	(M,2), (N,4), (O,1)
6	(Q,4), (Q,7), (P,3)
7	(R,2), (R,4), (R,5), (S,3), (S,6)

Theorem 3.1. *Given any instance I of the CDMD problem, the procedure just described generates a schedule with a TCT of at most $3.5d$ in $O(nq)$ time.*

3.1 Transformation from the $CDMD$ to the $CDMD_{MSD}$ Problem

Given any instance of the *CDMD* problem, we define the following terms. For every processor i, we define the set of *message-destination pairs* (*md-pairs* for short) of the form (message-id, processor index) that contains one entry for each message that processor i has to send to a different destination. Table 3 displays all the md-pairs originating at each processor for the instance given in Example 1.1. We corrupt our notation and refer to the length of the md-pair to mean the length of the message associated with the md-pair. For example, md-pairs $(A, 2)$ and $(A, 5)$ have length 35, and md-pair $(B, 4)$ has length 2. Two md-pairs are said to be *equivalent* if they correspond to the same message, and *nonequivalent* otherwise. For example, md-pairs $(A, 2)$ and $(A, 5)$ are equivalent, but md-pairs $(A, 2)$ and $(B, 4)$ are not.

An md-pair is labeled *long* if it has a length greater than $d/2$, and *short* otherwise. In our example, all the md-pairs are short, except for $(A, 2)$, $(A, 5)$, $(E, 1)$, $(H, 7)$, $(S, 3)$, and $(S, 6)$. Each processor may hold zero or more long md-pairs, but will not hold two nonequivalent long md-pairs; otherwise, the definition of d is contradicted. Though any processor may have two or more equivalent long md-pairs. However, the total number of long md-pairs is at most n. If there were more than n long md-pairs, then at least two of the md-pairs will have the same processor destination, and that processor will receive messages with a total length greater than d, which contradicts the definition of d. A processor with one or more long md-pairs is said to be of

TABLE 4
Type, t-Length, and r-Length of md-Pairs

	Original Instance							Resulting Instance						
Processor	1	2	3	4	5	6	7	1	2	3	4	5	6	7
t-length	141	46	48	9	32	12	122	81	46	48	69	72	12	82
r-length	106	46	48	9	32	12	91	81	46	48	69	72	12	51
Type	H_1	L_0	L_1	L_0	L_0	L_0	H_1	F_1	L_0	L_1	F_1	F_0	L_0	L_1

type 1; otherwise, it is of *type 0*. The *total length (or t-length)* of a set of md-pairs S is the sum of the length of the md-pairs in S. The *restricted length (or r-length)* of a set of md-pairs S is the same as the t-length, except that the equivalent long md-pairs will be included in the total r-length once, rather than once for each md-pair. The left portion of Table 4 lists the t-length and the r-length of md-pairs to be sent by the processors for the instance in Example 1.1.

A processor type i, for $0 \leq i \leq 1$, is said to be *light (L)*, *full (F)*, and *heavy (H)* if the r-length of the md-pairs that it must send is in the range $[0, d)$, $[d, 1.5d]$, and $(1.5d, \infty)$, respectively. We refer to processors as being of type L_i, F_i, and H_i to mean a type i processor, and L, F, and H indicate that the processor is light, full, or heavy. The left portion of Table 4 lists the type of the processors in our example. In what follows, we corrupt our notation and say that "an md-pair is being forwarded from processor i to processor j" to mean that the message corresponding to the md-pair is being forwarded from processor i to processor j, and that the md-pair is moved from processor i to processor j.

We perform the following transformation, which we call LH, on a processor l of type L_0 or L_1, and on a processor h of type H_0, or H_1, as long as such processors exist.

- [Operation 1] If processor h is of type H_1 and the md-pairs in processor l have a total r-length of at most $0.5d$, then forward a long md-pair from processor h to processor l.
- [Operation 2] As long as processor l remains light, forward a short md-pair from processor h to processor l.

Applying transformation LH to processor 1 of type H_1 and processor 4 of type L_0 results in the forwarding of the following md-pairs: $(A, 5)$, $(B, 4)$, and $(C, 6)$. Operation 1 forwards the first md-pair, and operation 2 forwards the other two. The application of transformation LH to processor 7 of type H_1 and processor 5 of type L_0 forwards the following md-pairs: $(R, 2)$ and $(R, 5)$. Fig. 1b shows the resulting problem instance (which we call the *resulting instance*), and the right portion of Table 4 gives the t-length, r-length, and type of each processor. The message multicasts include only those represented by long md-pairs. For example, the md-pairs in processor 1 for messages C will not be treated as one multicast, they will be treated as two different messages to be transmitted at different times. On the other hand, the two md-pairs for message S being sent from processor 7 will be treated as a multicast. Note that in the resulting problem instance, there are two md-pairs ($(B, 4)$ in P_4 and $(R, 5)$ in P_5) whose message needs to be sent to the processor in which they reside! Obviously, one does not need to send those messages, and the two md-pairs can be eliminated. In addition, message C does not really need to be sent from processor 1 to processor 4 since processor 4 already holds it because message C was forwarded to processor 4. This same situation arises in other problem instances, but the worst case values for s', r', and d' do not change when we eliminate these self-loops.

The possible transitions for the heavy processors, resulting from the application of transformation LH, are shown in Fig. 2. In the following lemma, we establish the possible transitions for the processors when applying transformation LH.

Lemma 3.1. *Transformation LH when applied to a processor l of type L_0 or L_1 and to a processor h of type H_0 or H_1 forwards a set of md-pairs from processor h to processor l in such a way that processor l becomes a full processor and processor h remains heavy or becomes a full or light processor.*

Proof. The first case is when processor h is of type H_1 and processor l has md-pairs with an r-length of at most $0.5d$.

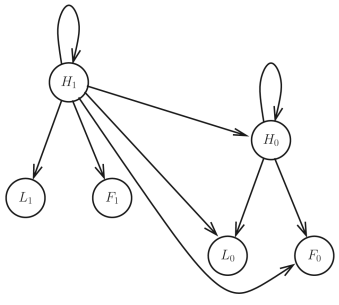

Fig. 2. Edges represent possible transitions resulting from the application of transformation LH to a processor of type H_0 or H_1.

In this case, Operation 1 forwards a long md-pair from processor h to processor l. If processor l remains light, then Operation 2 forwards short md-pairs from processor h to processor l until it becomes a full processor. The total r-length of the forwarded md-pairs is at most $1.5d$. Since the initial r-length of the h processor is more than $1.5d$, it follows that processor l will become a full processor. Processor h remains of type H_1 or becomes H_0, full (F_1, F_0), or light (L_1, L_0).

The second case is when processor h is of type H_0 and processor l has md-pairs with an r-length of at most $0.5d$. In this case, only Operation 2 is applied and forwards short md-pairs for processor h to processor l until processor l becomes full. The total r-length of the forwarded md-pairs is at most $1.5d$. Since processor h has initially short md-pairs with a length of more than $1.5d$, it follows that processor l will become full. On the other hand, processor h may remain heavy (H_0) or becomes full (F_0) or light (L_1, L_0).

The last case is when processor l has md-pairs with an r-length of more than $0.5d$. In this case, only Operation 2 is applied and forwards short md-pairs from processor h to processor l until processor l becomes full. The total r-length of the forwarded md-pairs is at most d. Since the h processor has initially short md-pairs with a total r-length of more than $0.5d$, it follows that processor l will become full. On the other hand, processor h may remain heavy (H_1, h_0) or becomes full (F_1, F_0) or light (L_1, L_0).

This concludes the proof of the lemma. □

Lemma 3.1 shows that every time that we apply transformation LH, the total number of full processors increases. Since once a processor becomes a full processor after the application of transformation LH, it remains a full processor throughout the remaining transformations. Therefore, the procedure terminates after at most n iterations.

We claim that at the time transformation LH is no longer applicable, every processor will be full or light; that is, none of the processors will be heavy. This follows from the fact that if there is at least one heavy processor and the remaining processors are heavy or full, then the length of the md-pairs in each of these processors will be at least d, and at least $1.5d$ for the heavy processors. It then follows that the total length of the md-pairs is more than dn. However, since every md-pair is to be received by a processor, this means that at least one processor must receive messages with a total length greater than d, contradicting the fact that every processor must receive md-pairs with a total length of at most d. Thus, it must be that when transformation LH can no longer be applied, all the processors will be light or full, and the resulting instance is an instance of the $CDMD_{MSD}$ problem with $s' \leq 1.5d$ and $r' \leq d$.

Theorem 3.2. *When transformation LH is no longer applicable to an instance of the CDMD problem, the resulting instance is an instance of the $CDMD_{MSD}$ problem, where $s' \leq 1.5d$ and $r' \leq d$. Furthermore, all the message forwarding can be carried out by a schedule with a TCT of at most d. The time complexity for the procedure is $O(q')$.*

Proof. The proof for the first statement follows from the above discussion. We now show that the forwarding can be carried out by a schedule with a TCT of d. Every processor that forwards messages will only forward messages that are in its initial hold set. All the messages sent to more than one destination will be multicasted. Therefore, the total forwarding time for each processor is at most d. Processors that receive forwarded messages receive the messages from exactly one processor. Therefore, these processors receive the messages at some points in time from time zero to time d. It then follows that all the message forwarding can be carried out by a schedule with a TCT of at most d. A careful implementation of the above transformation can be shown to take time proportional to q', which is the total number of messages received by the processors. One may implement the above procedure by accessing each md-pair a constant number of times. □

3.2 Algorithms for $CDMD_{SD}$ and $CDMD_{MSD}$

First, we discuss our algorithm for the $CDMD_{SD}$ problem and, then, the one for the $CDMD_{MSD}$ problem. A schedule for an instance of the $CDMD_{SD}$ problem is constructed as follows: Whenever a processor j finishes receiving a message at some time t (or at time zero, and it has not started receiving messages), we search for a processor that is not currently sending any messages and holds a message whose destination is processor j that has not yet been sent. If such processor exists, for example, processor k, the message is sent without interruption from processor k to processor j starting at time t. On the other hand, if no such processor can be identified, then processor j is not scheduled to receive a message starting at this time. In this case, it must be that all the processors that hold an unsent message whose destination is processor j are sending messages to other processors. Whenever a processor finishes sending a message, we will check again for a pair of processors with the above properties.

We claim that the communication schedule constructed by the above algorithm has a TCT of at most $s' + r'$; remember that s' and r' are the total length of the messages any processor may send and receive, respectively. The proof of this fact is simple. Let j be a processor that receives a message at the latest time t. In case of ties, select any processor that satisfies the property. Let us call this last message X. Let k be the processor that sent that last message X to processor j (see Fig. 3). Let us now examine processor j from time zero to time t. Clearly, at all times, it is either busy receiving messages (darker areas in Fig. 3) or idle [not receiving any messages (shaded regions in Fig. 3)]. Clearly, processor j cannot receive messages for more than r' time units. When processor j is idle (not receiving any messages), it must have been that some other processor must have been receiving a message that processor k was sending. Otherwise, it would contradict the way we construct our communication schedules because message X could have been sent to processor j at that time. Therefore, the total time processor j is idle is at most s', and the schedule constructed by the algorithm has a TCT of at most $s' + r'$.

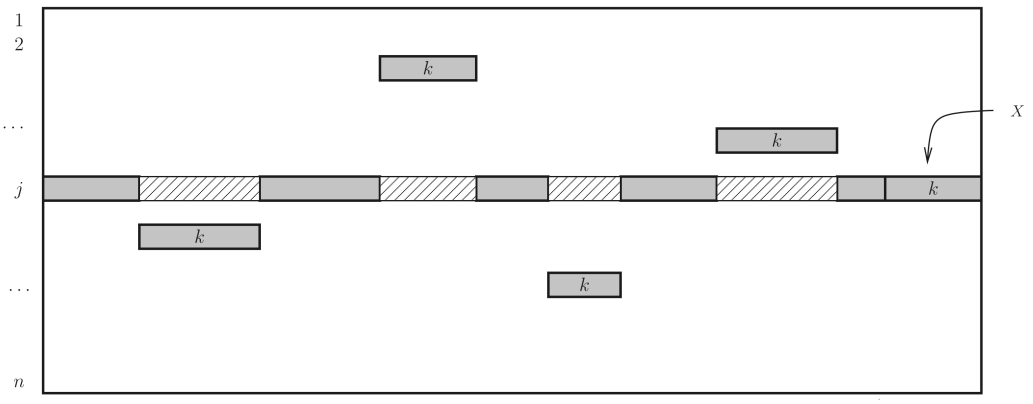

Fig. 3. Type of schedule constructed by Procedure LS. The vertical axis denotes processors, and the horizontal one denotes time.

Before we discuss implementation details, we apply the following preprocessing procedure to reduce the overall time complexity of our algorithm. Assume that the instance of the $CDMD_{SD}$ is such that there is at most one message with the same origin and destination. If this is not the case, we can combine all the messages with the same origin and destination into one. Once we generate a schedule for the resulting instance, it is simple to construct a schedule for the original problem instance. In the resulting instance, each processor sends at most $n' - 1$ messages, each to a different processor. The transformation can be easily implemented to take $O(q')$ time.

We represent each md-pair by the tuple (i, j, l) (meaning: send the message from processor i to processor j for l time units). The tuple contains other entries to be used when storing the tuple in a list or heap. For $1 \leq i \leq n'$ and $1 \leq j \leq n'$, if processor i sends a message to processor j, we add the tuple (i, j, l) to F-list(i), and T-list(j), where the F-list(i) has all tuples representing messages with origin processor i, and T-list(j) has all tuples representing messages with destination processor j. The lists T-rts(j) for $1 \leq j \leq n'$ will contain all the tuples (i, j, l) for md-pairs that represent messages that are currently ready to be sent to processor j because processor i is not currently sending any message. These T-rts(j) lists will be initialized in the first part of the procedure to include all tuples representing all the md-pairs. The heap called *next-to-finish* contains the tuples (i, j, l, t) corresponding to the md-pairs representing messages that are currently being transmitted. The heap is a min-heap with respect to the value of t, which is the time when the transmission of the message represented the md-pair corresponding to the tuple will finish. The algorithm operates as follows:

```
Procedure LS
 for each tuple (i,j,l), add (i,j,l) to list T-rts(j) endfor
 current.time = 0
 for j = 1 to n' do
  Invoke Procedure Process-T-rts(j) //the procedure
   is defined below//
 endfor
 while the heap next-to-finish is not empty do
  delete a tuple with minimum t value from next-to-finish
   heap
  let (i,j,l,t) be the tuple deleted
  current.time = t
  add each tuple from F-LIST(i) to its corresponding
   T-rts list
  for each idle processor j such that T-rts(j) is not
   empty do
   Invoke Procedure Process-T-rts(j)
  endfor
 endwhile
End Procedure LS

Procedure Process-T-rts(j)
  let (i,j,l,t) be the first tuple in list T-rts(j)
  add to the next-to-finish heap the tuple
   (i,j,l,current.time + l)
  delete (i,j,l) from F-list(i)
  delete (i,j,l) from T-list(j)
  delete all the tuples from list T-rts(j)
  for each tuple (i,j',l') in F-list(i) and in T-rts(j'), delete
   (i,j',l') from T-rts(j')
End Procedure Process-T-rts(j)
```

The initialization phase inserts q' tuples to the T-lists and F-lists. This takes $O(q')$ time. Every time procedure Process-T-rts is executed, it takes $O(n')$, provided the lists T-rts(j) and T-list(j) are implemented as doubly linked lists and enough pointers have been set to locate the tuples in these lists. Whenever this procedure is executed, one message represented by an md-pair will be scheduled.

The first loop in Procedure LS is executed n' times, so, in total, it takes $O(n' \log n')$ time. The second loop is executed q' times, once for each message-destination pair. The delete operation from the heap takes $O(\log n')$ time. The addition of the tuples to the list takes constant time, but it may be repeated n' times. The innermost while loop is checked n' times, but the check takes constant time. The total number of times that Procedure Process-T-rts is invoked is q'. Therefore, the time complexity for the second loop is $O(q'n')$ time.

Theorem 3.3. *Given any instance I of the $CDMD_{SD}$ problem, the procedure LS described above generates a schedule with a TCT of at most $s' + r'$. Furthermore, the time complexity is $O(n'q')$.*

Proof. By the above discussion. □

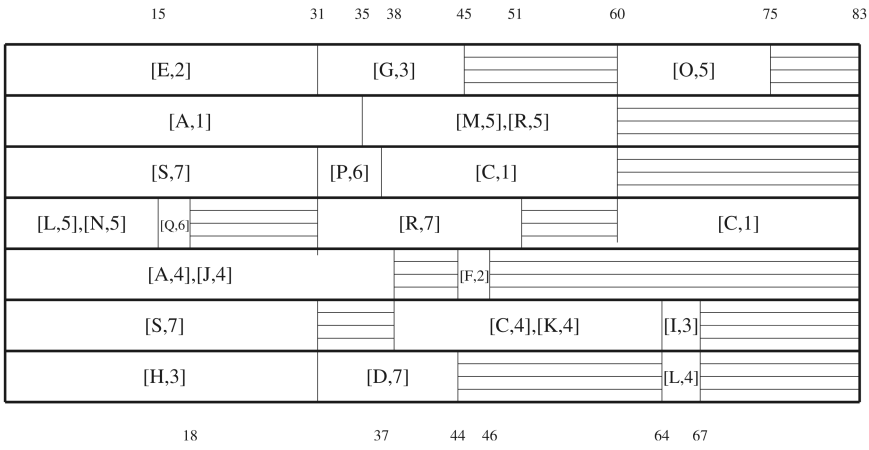

Fig. 4. Schedule for the resulting instance constructed from Example 1.1.

The $CDMD_{SD}$ problem can be viewed as the problem of minimizing the makespan for scheduling a set of jobs without preemptions in an open shop. In fact, a scheduling algorithm is a version of the list schedule developed for the open-shop problem by Racsmány. The analysis is similar to the one developed by Shmoys et al. [26] for open-shop list schedules.

We claim that the above procedure also works for the $CDMD_{MSD}$, provided that at time zero, one sends all the multidestination messages. Since all of them have different destinations, there will be no conflict. Once we do this, we continue (perhaps scheduling at time zero on idle processors) scheduling the tasks as Procedure LS. It is simple to prove the following result:

Theorem 3.4. *Given any instance I of the $CDMD_{MSD}$ problem, the modified version of procedure LS described above generates a schedule with a TCT of at most $s' + r'$ in $O(n'q')$ time.*

Fig. 4 depicts a schedule generated by procedure LH for the resulting problem instance of the instance given in Example 1.1 (Fig. 1b with the edges for md-pairs $(B, 4)$ and $(R, 5)$ deleted, but keeping the edge for the md-pair $(C, 4)$ in P_1). The portions of the schedule with multiple horizontal lines correspond to periods of idle time; that is, the processors are not receiving messages. The pairs inside square brackets are not md-pairs. Their first component is the message, and the second one is the processor where it came from. For example, the pair $[E, 2]$ represents the md-pair $(E, 1)$ in processor 2. Note that at any given time, all the messages being transmitted originate at different processors. The only exception is at time zero when large messages are being multicasted (message S sent to processors 3 and 6). Note that in the schedule given in Fig. 4, you will find blocks of time where two pairs are scheduled (for example, processors 4 and 5 at time zero). These pairs represent messages that originate at the same processor (i) and have the same destination $(j \neq i)$. In the preprocessing procedure, they were combined into one.

4 Discussion

We presented an approximation algorithm for the *CDMD* problem over the n-processor single-port complete (or fully connected) static network with the multicasting communication primitive. Our algorithm constructs a message-routing schedule with a TCT of at most $3.5d$ for every degree d problem instance, where d is the total length of the messages that each processor may send (or receive). Remember that d is a lower bound for the TCT of any communication schedule. Therefore, the approximation ratio of our algorithm is 3.5. Our algorithm takes $O(nq)$ time, where n is the number of processors and q is the total number of message-destination pairs.

There remain several interesting open problems. The first one is the development of an algorithm for the $CDMD_{MSD}$ problem with an approximation ratio smaller than $s' + r'$ (Theorem 3.4). A large class of greedy heuristics for the $CDMD_{MSD}$ problem can be shown to have the same approximation ratio as the modified procedure LS used in Theorem 3.4. Any approximation algorithm for the $CDMD_{SD}$ problem is also an approximation for the minimum makespan nonpreemptive open-shop problem. Shmoys et al. [26] suggest a greedy algorithm for the open-shop problem where a task of a job with the largest remaining time is scheduled first. However, this and other variations have been shown to have the same approximation ratio of 2. Williamson et al. [27] have shown that it is NP-hard to approximate the makespan nonpreemptive open-shop problem within $\frac{5}{4}$. However, this result holds when the makespan of an optimal schedule is 4. It may be possible to asymptotically approximate the makespan open-shop problem below $\frac{5}{4}$ without implying that $P = NP$.

Our approximation algorithms for the *CDMD*, as well as the multimessage multicasting problems, fall into the same pattern. Perform a set of message-forwarding operations, and then, solve optimally or suboptimally the resulting restricted routing problem. It is not clear whether a different approach can be used to generate equivalent approximation algorithms. This is an intriguing open problem. If, indeed, it is possible to do so, it might provide the key to generate closer to optimal solutions to a large class of message dissemination problems.

REFERENCES

[1] T.F. Gonzalez, "Complexity and Approximations for Multimessage Multicasting," *J. Parallel and Distributed Computing,* vol. 55, no. 2, pp. 215-235, 1998.

[2] T.F. Gonzalez, "Simple Multimessage Multicasting Approximation Algorithms with Forwarding," *Algorithmica,* vol. 29, pp. 511-533, 2001.

[3] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 11, pp. 1143-1156, 1997.

[4] A. Goldman, J.G. Peters, and D. Trystram, "Exchanging Messages of Different Sizes," *J. Parallel and Distributed Computing,* vol. 66, pp. 1-18, 2006.

[5] S. Ranka, R.V. Shankar, and K.A. Alsabti, "Many-to-Many Personalized Communication with Bounded Traffic," *Proc. Fifth Symp. the Frontiers of Massively Parallel Computation (Frontiers '95),* pp. 20-27, 1995.

[6] Y.-J. Suh and K.G. Shin, "All-to-All Personalized Communication in Multidimensional Torus and Mesh Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 1, pp. 38-55, 2001.

[7] S. Ranka, J.-C. Wang, and G.C. Fox, "Static and Run-time Algorithms for All-to-Many Personalized Communications on Permutation Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 12, pp. 1266-1274, Dec. 1994.

[8] E.J. Coffman Jr., M.R. Garey, D.S. Johnson, and A.S. LaPaugh, "Scheduling File Transfers in Distributed Networks," *SIAM J. Computing,* vol. 14, no. 3, pp. 744-780, 1985.

[9] J. Whitehead, "The Complexity of File Transfer Scheduling with Forwarding," *SIAM J. Computing,* vol. 19, no. 2, pp. 222-245, 1990.

[10] H.A. Choi and S.L. Hakimi, "Data Transfers in Networks," *Algorithmica,* vol. 3, pp. 223-245, 1988.

[11] B. Hajek and G. Sasaki, "Link Scheduling in Polynomial Time," *IEEE Trans. Information Theory,* vol. 34, no. 5, pp. 910-917, 1988.

[12] I.S. Gopal, G. Bongiovanni, M.A. Bonuccelli, D.T. Tang, and C.K. Wong, "An Optimal Switching Algorithm for Multibeam Satellite Systems with Variable Band Width Beams," *IEEE Trans. Comm.,* vol. 30, no. 11, pp. 2475-2481, 1982.

[13] P.I. Rivera-Vega, R. Varadarajan, and S.B. Navathe, "Scheduling File Transfers in Fully Connected Networks," *Networks,* vol. 22, pp. 563-588, 1992.

[14] J. Hall, J. Hartline, A.R. Karlin, J. Saia, and J. Wilkes, "On Algorithms for Efficient Data Migration," *Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '01),* pp. 620-629, 2001.

[15] T.F. Gonzalez, "An Efficient Algorithm for Gossiping in the Multicasting Communication Environment," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 7, pp. 701-708, July 2003.

[16] T.F. Gonzalez, "Multimessage Multicasting," *Proc. Irregular '96,* pp. 217-228, 1996.

[17] T.F. Gonzalez, "Distributed Multimessage Multicasting," *J. Interconnection Networks,* vol. 1, no. 4, pp. 303-315, 2000.

[18] T.F. Gonzalez, "Message Dissemination Using Modern Communication Primitives," *Handbook Parallel Computing: Models, Algorithms, and Applications,* S. Rajasekaran and J. Reif, eds., Chapman & Hall/CRC, chapter 36, 2008.

[19] H. Shen, "Efficient Multiple Multicasting in Hypercubes," *J. Systems Architecture,* vol. 43, no. 9, pp. 655-662, 1997.

[20] D. Thaker and G. Rouskas, "Multi-Destination Communication in Broadcast WDM Networks: A Survey," *Optical Networks,* vol. 3, no. 1, pp. 34-44, 2002.

[21] S. Khuller, Y.-A. Kim, and Y-C. Wan, "Algorithms for Data Migration with Cloning," *SIAM J. Computing,* vol. 33, no. 2, pp. 448-461, 2004.

[22] S. Khuller, Y.-A. Kim, and Y.-C. Wan, "Broadcasting on Networks of Workstations," *Proc. 17th ACM Symp. Parallelism in Algorithms and Architectures (SPAA),* 2005.

[23] R. Gandhi, M.M. Halldorsson, M. Kortsarz, and H. Shachnai, "Improved Results for Data Migration and Open Shop Scheduling," *ACM Trans. Algorithms,* vol. 2, no. 1, pp. 116-129, 2006.

[24] Y.S. Hwang, R. Das, J. Saltz, M. Hodoscek, and B. Brooks, "Parallelizing Molecular Dynamics Programs for Distributed Memory Machines," *IEEE Comput. Sci. Eng.,* vol. 2, no. 2, pp. 18-29, 1995.

[25] T.F. Gonzalez, "Continuous Message Dissemination under the Multicasting Mode," *Proc. 18th IASTED Int'l Conf. Parallel and Distributed Computing and Systems (PDCS),* 2006.

[26] D.B. Shmoys, C. Stein, and J. Wein, "Improved Approximation Algorithms for Shop Scheduling Problems," *SIAM J. Computing,* vol. 23, pp. 617-632, 1994.

[27] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevast'janov, and D.B. Shmoys, "Short Shop Schedules," *Operations Research,* vol. 45, no. 2, pp. 288-294, 1997.

Teofilo F. Gonzalez received the BSc degree in computer science from the Instituto Tecnologico de Monterrey, Monterrey, Mexico (1972) and the PhD degree in computer science from the University of Minnesota, Minneapolis (1975). He is currently a professor of computer science at the University of California, Santa Barbara. His research activity has concentrated on the development of efficient exact and approximation algorithms for problems in several areas, including job scheduling, message dissemination in parallel and distributed systems, CAD/VLSI placement and routing, and graph applications. His current research interests include multimessage multicasting algorithms, routing algorithms, and approximation algorithms. He has served on the editorial board of several publications and as program committee chair for several conferences. He is the editor of the *Handbook of Approximation Algorithms and Metaheuristics* (2007). He is a member of the IEEE.