Proceedings of the 18th IASTED International Conference
Parallel and Distributed Computing and Systems
November 13–15, 2006, Dallas, TX, USA
ISBN Hardcopy: 0-88986-638-4 / CD: 0-88986-640-6

CONTINUOUS DELIVERY MESSAGE DISSEMINATION PROBLEMS UNDER THE MULTICASTING COMMUNICATION MODE

Teofilo F. Gonzalez
Department of Computer Science
University of California
Santa Barbara, CA, 93106, USA
email: teo@cs.ucsb.edu

ABSTRACT
We consider the continuously delivery message dissemination ($CDMD$) problem under the multicasting communication mode over the n processor complete (all links are present and are bi-directional) static networks. 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 total communication time at most $3.5d$, where d is the total length of the messages that each processor may send or receive. Our algorithm takes $O(m^2 + n \log n)$ time, where n is the number of processors and m is the number of messages.

KEY WORDS
Approximation Algorithms, Continuous Delivery, Message Dissemination, Multicasting Communication, 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 a 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 perform all the intertask communications efficiently. 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. Given a network with a communication model, a set of communication primitives and a set of messages that need to be exchanged, our problem 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, the messages have different length, but may be partitioned into packets. However, every message must arrive to its destination in its "original" order and all its packets must arrive during consecutive time units. One may think of the messages as "video clips" to be viewed without delay on arrival or data that needs to be processed on-line in the order it is generated. Generating an optimal communication schedule, i.e., one with the least total communication rounds, for the $CDMD$ problem, even when all the messages consist of just one packet, 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. In this paper we consider the continuous delivery message dissemination ($CDMD$) problem for n processor complete networks. For the case when the messages have arbitrary length, we present an efficient approximation algorithm to construct a message routing schedule with total communication time at most $3.5d$, where d is the total length of the messages that each processor may send (or receive). Our algorithm takes $O(m^2 + n \log n)$ time, where n is the number of processors and m is the number of messages.

The $CDMD$ problem consists of constructing a communication schedule, for an n processor complete static (all links are present and are bi-directional) network N, with least total communication time for multicasting (transmitting) any given set of messages. Specifically, there are n processors, $P = \{P_1, P_2, \ldots, P_n\}$, interconnected via network N. Each processor needs to send a set of messages each with a possibly different length and to a nonempty subset of processors. The messages must travel through the network. Our objective is to determine when each of these messages is to be transmitted so that all the communications can be carried in the least total amount of time.

Let us formally define our problem. Each processor P_i *holds* the set of messages h_i each with a possibly different length (positive integer) and *needs* to receive the set of messages n_i. Message j has length l_j (positive integer) which is the number of its packets or time units needed to transmit the message from one processor to another. The messages may be partitioned into units or packets, but every processor must receive all the packets of each message in order and during consecutive time units. 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 any processor sends or *max hold* of a problem instance as $s = \mathbf{max}\{\sum_{j \in h_i} \mid l_j \mid\}$. Similarly, the total length of

the messages any processor receives or the *max need* of a problem instance as $r = \mathbf{max}\{\sum_{j \in n_i} \mid l_j \mid\}$. We define the *degree* of a problem instance as $d = \mathbf{max}\{s, r\}$, i.e., the maximum total length of the messages any processor sends or receives. We use m to denote the total number of messages. Consider the following example.

Example 1.1 *There are seven processors ($n = 7$) all of which need to send and receive messages and the total number of messages $m = 18$. The messages each processor holds and needs are given in Table 1. The length (in packets or time units) of the messages are given in Table 2. For this example $d = s = r = 60$.*

Table 1. Hold and Need vectors for Example 1.1.

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

h_5	h_6	h_7
$\{L, M\}$	$\{N, O\}$	$\{P, Q, R\}$
n_5	n_6	n_7
$\{A, F, J, P\}$	$\{C, I, K, R\}$	$\{D, H, M, O\}$

Table 2. Message Lengths for Example 1.1.

A	B	C	D	E	F	G	H	I
30	10	20	5	40	10	20	35	5
J	K	L	M	N	O	P	Q	R
10	5	20	10	20	10	10	30	20

Usually one visualizes these type of problems by directed multigraphs. Each processor P_i is represented by the vertex labeled i, and each message is represents 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 Figure 1 as a directed multigraph with additional thick lines that identify all edges or branches in each bundle.

In the *single port communication mode* every processor sends at most one message and receives at most one message during each communication round. A processor may send at each communication round 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. This is why we call this communication mode *multicasting*. The message also remains in the

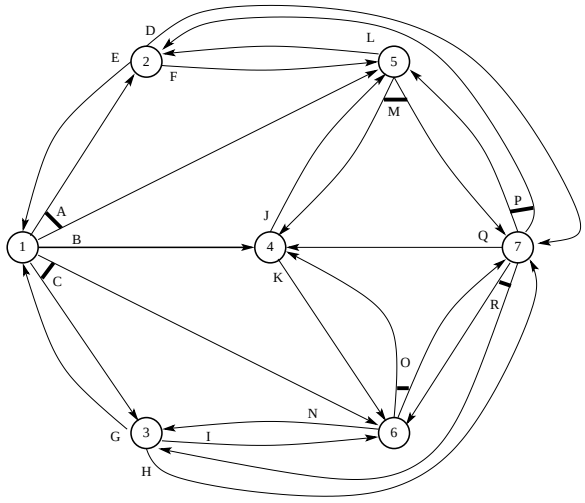

Figure 1. Example for 1.1. The thick lines joins all the edges (branches) in the same bundle.

hold set h_i. During each time unit 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 communication round.

The communication process ends when each processor has $n_i \subseteq h_i$, i.e., each processor holds all the messages it needs. The *total communication time* (or simply *TCT*) is the total number of communication rounds. Our communication model allows us to transmit any of the messages to only a subset of its destinations at a time. I.e., any given message may be transmitted at different times. This added routing flexibility reduces the TCT. In many cases it is a considerable reduction.

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

2 Applications and Previous Results

The $CDMD$ problem arises in applications where a non-preemptive message exchange solution is desired. For example when the messages are video clips being exchanged by servers in order to better satisfy users requests [2].

A restricted version of the $CDMD$ problem where all messages have have equal length and each message has only one destination is called the MU_C problem. The MU_C problem and its variants have been studied. The basic results include: heuristics, approximation algorithms, polynomial time algorithms for restricted versions of the problem, and NP-completeness results. Coffman *et al.*

[3] present approximation algorithms for a generalization where there are $\gamma(P_i)$ sending or receiving ports per processor. The power of message forwarding was reported by Whitehead [4]. When preemptions are allowed, messages may be transmitted with interruptions, generalizations of the problem have been considered by Choi and Hakimi [5], Hajek and Sasaki [6], Gopal *et al.* [7]. The n-port MU_C problem over complete networks, for transferring files, was studied by Rivera-Vega *et al.* [8]. A variation of the problem, called *the message exchange problem*, has been studied by Goldman *et al.* [9]. The communication model in this version of the problem is asynchronous and it closely corresponds to actual distributed memory machines. Another restricted version of the MU_C problem, called *data migration* was studied by Hall *et al.* [10].

The main difference between the above research and the one we discuss in this paper is that we concentrate on the multicasting communication mode, rather than on the *telephone communication mode*. Most of the previous research based on the multicasting communication mode has concentrated on single messages. The exception is the work on the multimessage multicasting problem, but it involves equal length messages [1, 11, 12, 13, 14, 15, 16, 17]. The initial research by Shen [16] 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. More recently, different strategies for solving multimessage multicasting problems over all-optical networks were surveyed by Thaker and Rouskas [17]. 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.* [18]. The TCT for optimal schedules under the telephone communication mode is considerably larger than the one over the multicasting communication mode. A generalization of this message dissemination problem allowing for limited broadcasting or multicasting was considered by Khuller *et al.* [2]. 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.* [19]. This version of the message dissemination problem is 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 most dynamic programming procedures in a parallel or distributed computing environment. In information systems, these problems arise naturally when multicasting information over a b-channel ad-hoc wireless communication network. Other applications include sorting, matrix multiplication, discrete Fourier transform, etc. Message routing problems under the multicasting communication primitives arise in sensor networks which are simply static or slow changing ad-hoc wireless networks. These type of networks have 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 increase rapidly in the near future. The nonpreemptive scheduling mode in our problems has additional applications when the cost of preemptions is large.

Gonzalez [1] shows that even restricted versions of the multimessage multicasting problem are NP-complete, but schedules with TCT at most d^2 can be constructed in polynomial 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 problem remains NP-complete, but schedules with TCT at most $2d$ can be constructed in $O(m^2+n\log n)$ time [11].

3 New Results

In this section we present our approximation algorithm for the $CDMD$ problem under the multicasting communication mode. We assume a fully connected interconnection network with bi-directional links between every pair of processors. We show that in polynomial time it is always possible to construct a schedule with TCT at most $3.5d$. The difference from previous versions of the problem studied is that messages have different length and even though messages can be split into packets all the packets for the same message must arrive in order and during consecutive time units to their destinations. But the same message may arrive at different times to different processors.

The *single destination* continuous delivery message dissemination $CDMD_{SD}$ problem is the $CDMD$ such that every message is sent to only one destination. The $CDMD_{MSD}$ problem is a slight variation of the $CDMD_{SD}$ problem where every processor may have one multidestination message (1 multicast). However all the multidestination messages (originating at all the processors) have different destinations. In Section 3.1 we show how to construct a schedule with TCT $s + r$ for every instance of the $CDMD_{MSD}$ problem. Then in Section 3.2 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 $s = 1.5d$ and $r = d$. The algorithm takes $O(m^2 + n\log n)$ time and consists of a message multicasting forwarding step. We show that this forwarding operation can be carried out by a

schedule S with TCT d. Furthermore, schedule S followed by any schedule for the instance $f(I)$ of the $CDMD_{MSD}$ is a schedule for the instance I of the $CDMD$ problem. Any schedule constructed in this fashion has TCT at most $3.5d$ (the first part of the schedule has TCT at most d and the second part has TCT at most $2.5d$).

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

Our algorithm for any instance of the $CDMD_{SD}$ problem is simple. It falls in the broad category of list schedules. These schedules are generated as follows. You are given an ordered list L of the processors. Then whenever a processor j finishes receiving a message (or has not started receiving one), we find the first processor k in the list that is not currently sending any messages and it holds a previously unsent message with destination processor j. This message (if it exists) will be sent without interruption. On the other hand if no processor k can be identified then processor j will not receive messages until another processor terminates sending a message. At that time we will check again if a processor k with the above properties exists. Note that the list is not actually necessary, and one could do the assignment in any order. The only purpose for the list is to break ties. Though, ties could be broken in any arbitrary order.

We claim that the list schedule constructed by the above algorithm has TCT at most $s + r$, remember that s and r are the total length of the messages any processor may send and received, respectively. The proof of this fact is straight forward. Let j be a processor that receives a message at the latest time. In case of ties, select any processor that satisfies the property. Lets call this last message X. Let k be the processor that sent message X to processor j. Let us now examine processor j time zero to time t. Clearly, at all times it was either busy receiving messages or idle (not receiving any messages). Since processor j cannot receive messages for more than r time units, then it received messages for at most r time units. When processor j was 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 list 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. So the schedule has TCT at most $s+r$. By using Fibonacci heaps one can show that the schedule can be constructed in $O(m^2 + n \log n)$ time.

Theorem 3.1 *Given any instance I of the $CDMD_{SD}$ problem the list schedules described above generates a schedule with TCT at most $s + r$. Furthermore the time complexity is $O(m^2 + n \log n)$.*

Proof: By the above discussion.

□

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, our list schedule corresponds to the list schedule developed by Racsmány discussed in [15].

In the next section we construct an instance of a slight variation of the $CDMD_{MSD}$ problem. We claim that the above procedure also works for the $CDMD_{MSD}$ provided that at time zero we send all the multidestination messages. Since all of them have different destinations there will not be a conflict. Once we do this we continue (perhaps scheduling at time zero on some processors) scheduling the tasks as the above algorithm. It is simple to prove the following result.

Theorem 3.2 *Given any instance I of the $CDMD_{MSD}$ problem the modified list schedules described above generates a schedule with TCT at most $s + r$ in $O(m^2 + n \log n)$ time*

In the next section we show how to construct from an instance of the $CDMD$ an instance of the $CDMD_{MSD}$ problem. The concatenation of the schedule in the next section and the list schedule is a schedule for the instance of the $CDMD$ problem.

3.2 Transformation from the $CDMD$ to the $CDMD_{MSD}$ problem.

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 sent by processor i to a different destination. For example processor 1 in Example 1.1 will have five message destination pairs: Two for message A ($(A, 2)$, $(A, 5)$), one for message B ($(B, 4)$), and two for message C ($(C, 6)$, $(C, 3)$). 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. Also we say that two md-pairs are *equivalent* if they correspond to the same message, and *non-equivalent* otherwise.

An md-pairs will be labeled *large* if it has length greater than $d/2$, and *short* otherwise. Each processor may have zero or more large md-pairs, but no processor will have two non-equivalent large md-pairs, as otherwise we contradict the definition of d. Though, any processor may have two or more equivalent large md-pairs. However, there can be at most n large md-pairs. The reason for this is that all of the md-pairs will be received by the processors and each processor will receive a set of md-pairs with total length at most d. If there were $n + 1$ large md-pairs, then at least two of the md-pairs will have the same destination and such processor will receive md-pairs with total length greater than d which contradicts the definition of d. A processor with zero, one, or more than one large md-pairs is said to be of *type 0, 1* or *2*, respectively.

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 total

length at most d. If we could do this then the resulting problem has a schedule with TCT d and the forwarding could be done by a schedule with TCT at most d. This will result in a schedule for the whole problem with TCT at most $2d$. However this is not always possible and even just determining whether or not this is possible is an NP-complete problem (bin packing) in the strong sense.

Because of this we need to take a more pragmatic approach. The idea is to forward a set of messages by using a schedule with TCT d, but the resulting problem will have that each processor sends a set of messages with total length at most $1.5d$ when counting the large equivalent md-pairs on a processor as single ones. The resulting problem is an instance of the $CDMD_{MSD}$.

A set or group of md-pairs is said to be *full* if the total length of the md-pairs in it is in the range $[d, 1.5d]$. Note that for this summation we count once all the equivalent md-pairs on the same processor. A processor type-i, for $0 \leq i \leq 2$, is said to be *light*, *full* and *heavy*, if the total length of the md-pairs in it is in the range $[0, d)$, $[d, 1.5d]$, and $(1.5d, \infty)$, respectively. We refer to processors as of type 0l, 0f, 0h, 1l, 1f, 1h, 2l, 2f or 2h to mean the type and length level of the processor.

We perform the following transformations until they can no longer be applied.

F1: Given a processor type-0l and one type-0h we forward a set of md-pairs from the type-0h processor to the type-0l processor. As a result of this we end up with a type-0f processor and the other being type-0h, type-0f, or type-0l.

F2: Given a processor type-1h and one type-0l with total length of md-pairs greater than or equal to $0.5d$, we forward small md-pairs from the processor type-1h to the processor type-0l until it becomes a full one. As a result of this transformation we end up with a type-0f processor and the other being type-1h, type-1f, or type-1l.

F3: Given a processor type-1h and one type-0l with total length of md-pairs less than $0.5d$, we forward the large md-pair and enough small md-pairs from the processor type-1h to the processor type-0l. As a result of this transformation we end up with a type-1f processor and the other being type-0h, type-0f, or type-0l.

F4: Given a processor type-2h and one type-0l with total length of md-pairs greater than or equal to $0.5d$, we forward small md-pairs from the processor type-2h to the processor type-0l until it becomes a full one. As a result of this transformation we end up with a type-0f processor and the other being type-2h, type-2f, or type-2l.

F5: Given a processor type-2h and one type-0l with total length of md-pairs less than $0.5d$, we forward a large md-pair and enough small md-pairs from the processor type-2h to the processor type-0l. As a result of this transformation we end up with a type-1f processor and the other being type-2h, type-2f, type-2l, type-1h, type-1f, or type-1l.

Every time that we apply the above transformations we increase the total number of full processors (i.e., processors type-0f, type-1f and type-2f). If the application of the above transformations eliminates all the heavy processors (i.e., type-0h, type-1h and type 2-h) then we have an instance of the desired $CDMD_{MSD}$ problem. On the other hand if we still have heavy processors then there are no type-0l processors. In this case we need to apply the following additional transformation.

We apply the following procedure to every processor independently. Sort all the small md-pairs in increasing (actually non-decreasing) order of their length. We will form a group of md-pairs for each processor that includes all the large md-pairs and as many of the small pieces until one md-pair is added and the group becomes full. As a result of this operation we will end with the *stay-here* group for the processor plus zero or more remaining small md-pairs. If the total length of the remaining small md-pairs is at most $0.25d$, then they can be added to the stay-here group and there will be no remaining small md-pairs. So, if there are remaining small md-pairs their total length is more than $0.25d$. The remaining md-pairs will be have to be forwarded to other processors, whereas the ones in the stay-here group will remain in the processor.

A processor whose stay-here group is light is said to be of type A and a processor with remaining small md-pairs is said to be of type X. Suppose processor a is type A and processor x is type X. Let q_a be the length of the md-pairs in the stay-here group of processor a, and q_x be the total length of the remaining md-pairs in processor x. We will forward a set of the small messages from processor x to processor a to be included in the stay-here group as follows. There are two cases depending on $q_a + q_x$.

- $q_a + q_x \geq d$: In this case we start forwarding the remaining small md-pairs from processor x to the stay-here group in processor a until the group becomes full. If the remaining small md-pairs in processor x have total length at most $0.25d$, they are also added to the group in processor a. We repeat the above process selecting a pair of processors type A and X. On the other hand, if the remaining small pieces in processor x have total length at least $0.25d$, then we select another processor type A and repeat the above procedure.

- $q_a + q_x < d$: In this case we add all the md-pairs from processor x to the stay-here group in processor a. We select another type X processor and repeat the procedure again.

We apply the above procedure until it cannot be further applied because there are no more processors of type A and/or X. It cannot be that there remain type X processors because then all processors will have md-pairs with total

length greater than d and it would contradict the fact that the total length of the md-pairs is at most dn. So it must be that there are no type X processors left. In this case the procedure terminates and we have created a problem instance of the $CDMD_{MSD}$ problem.

Theorem 3.3 *The above procedure constructs an instance of the $CDMD_{MSD}$ problem in which $s = 1.5d$ and $r = d$. Furthermore, all the message forwarding can be carried out in a schedule with TCT d.*

Proof: The proof for the first statement follows from the above discussion. The claim that all the forwarding can be carried by a schedule wit TCT d is more elaborate. This requires to specify the time at which each message is forwarded. For brevity we do not include the details, though they will be included in the full version of the paper.

□

4 Discussion

We presented an approximation algorithm for the continuously delivery message dissemination ($CDMD$) problem under the multicasting communication mode over the n processor complete (or fully connected) static networks. For the case when the messages have different length, we present an efficient approximation algorithm to construct a message routing schedule with TCT 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). Our algorithm takes $O(m^2 + n \log n)$ time, where n is the number of processors and m is the number of messages.

Acknowledgments

We like to thank the referees for pointing out mistakes in an arlier version of our paper and for their suggestions on ways to improve the readability of our paper.

References

[1] Gonzalez, T. F., Complexity and Approximations for Multimessage Multicasting, *J. of Parallel and Distributed Computing,* 55(2), 215, 1998.

[2] Khuller, S., Kim, Y.-A., and Wan, Y-C., Broadcasting on Networks of Workstations, *Proc. of SPAA*, 2005.

[3] Coffman, Jr. E. J., M. R. Garey, D. S. Johnson, and A. S. LaPaugh, Scheduling File Transfers in Distributed Networks, *SIAM J. on Computing*, 14(3), 744, 1985.

[4] Whitehead, J. The Complexity of File Transfer Scheduling with Forwarding, *SIAM J. on Computing,* 19(2), 222, 1990.

[5] Choi, H. A., and S. L. Hakimi, Data Transfers in Networks, *Algorithmica*, 3, 223, 1988.

[6] Hajek, B., and G. Sasaki, Link Scheduling in Polynomial Time, *IEEE Transactions on Information Theory*, 34(5), 910, 1988.

[7] Gopal, I. S., 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 Transactions on Communications*, 30(11), 2475, 1982.

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

[9] Goldman, A., Peters, J. G., and Trystram, D., Exchanging messages of different sizes, *J. of Par. and Dist. Comput,* 66, 18, 2006.

[10] Hall, J., Hartline, J., Karlin, A.R., Saia, J., and Wilkes, J., On Algorithms for Efficient Data Migration, *Proc. of SODA*, 620, 2001.

[11] Gonzalez, T. F., Simple Multimessage Multicasting Approximation Algorithms With Forwarding, *Algorithmica*, 29, 511, 2001.

[12] Gonzalez, T. F., MultiMessage Multicasting, Proceedings of Irregular'96, LNCS (1117), Springer, 217, 1996.

[13] Gonzalez, T. F., Improved Approximation Algorithms for Multimessage Multicasting, *Nordic Journal on Computing,* 5, 196, 1998.

[14] Gonzalez, T. F., Distributed Multimessage Multicasting, *Journal of Interconnection Networks,* 1(4), 303, 2000.

[15] Gonzalez, T. F., Message Dissemination Using Modern Communication Primitives, *Handbook Parallel Computing: Models, Algorithms, and Applications,* S. Rajasekaran and J. Reif Eds., CRC Press, (to appear).

[16] Shen, H, Efficient Multiple Multicasting in Hypercubes, *J. of Systems Architecture,* 43(9), 1997.

[17] Thaker, D. and Rouskas, G., Multi-Destination Communication in Broadcast WDM Networks: A Survey, *Optical Networks*, 3(1), 34, 2002.

[18] Khuller, S., Kim, Y.-A., and Wan, Y-C., Algorithms for Data Migration with Cloning, *SIAM J. Comput.*, 33(2), 448, 2004.

[19] Gandhi, R., Halldorsson, M.M., Kortsarz, M., and Shachnai, H., Improved Results for Data Migration and Open Shop Scheduling, *ACM Trans. on Algorithms,* 2(1), 116, 2006.