Algorithms for Multimessage Multicasting with Forwarding

T. F. Gonzalez
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

We consider Multimessage Multicasting over the n processor complete (or fully connected) static network (MM_C) when Forwarding of messages is allowed. We present an efficient algorithm that constructs for every degree d problem instance a schedule with total communication time at most $2d$, where d is the maximum number of messages that each processor may send (receive). Our centralized algorithms require all the communication information ahead of time. Applications where this information is available include iterative algorithms for solving linear equations, and most dynamic programming procedures. Our schedules can also be used by systems communicating via dynamic permutation networks, e.g., Meiko CS-2.

1 Introduction

The Multimessage Multicasting problem over the n processor static network (MM_C) consists of constructing a schedule 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 a network N. Each processor is executing processes, and these processes are exchanging messages that must be routed through the links of N. 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. To generate communication schedules with significantly smaller total communication time we allow *forwarding*, which means that messages may be sent through indirect paths even though single link paths exist! This version of the multicasting problem is referred to as the MMF_C problem.

Routing in the complete static network (there are bidirectional links between every pair of processors) is the simplest and most flexible when compared to other static and dynamic networks. Dynamic networks that can realize all Permutations and Replicate data (e.g., n by n Benes network based on 2 by 2 switches that can also act as data replicators) will be referred to as *pr-dynamic networks* (e.g., IBM GF11, and Meiko CS-2). Multimessage Multicasting for pr-dynamic and complete networks is not too different, in the sense that any schedule for a complete network can be translated automatically into an equivalent communication schedule for any pr-dynamic network. This is accomplished by translating each communication phase for the complete network into no more than two communication phases for the pr-dynamic networks. The first phase replicates data and transmits it to other processors, and the second phase distributes data to the appropriate processors ([10], [11], [14]).

Let us formally define our problem. Each processor P_i *holds* the set of messages h_i and *needs* to receive the set of messages n_i. We assume that $\bigcup h_i = \bigcup n_i$, and that each message is initially in exactly one set h_i. We define the *degree* of a problem instance as $d = \mathbf{max}\{h_i, n_i\}$, i.e., the maximum number of messages that any processor sends or receives.

Example 1.1 *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 $h_1 = \{a, b\}$, $h_2 = \{c, d\}$, $h_3 = \{e, f\}$, $h_4 = \ldots = h_9 = \emptyset$, $n_1 = n_2 = n_3 = \emptyset$, $n_4 = \{a, c, e\}$, $n_5 = \{a, d, f\}$, $n_6 = \{b, c, e\}$, $n_7 = \{b, d, f\}$, $n_8 = \{c, d, e\}$, and $n_9 = \{c, d, f\}$. The density is 3.*

One can 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 1.1 is depicted in Figure 1 as a directed multigraph with additional thin lines that identify all edges or branches in each bundle.

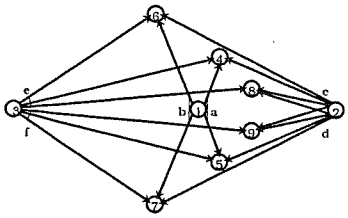

Figure 1: Directed Multigraph Representation for Example 1.1. The thin line joins all the edges (branches) in the same bundle.

The communications allowed in our complete network satisfy the following two restrictions.

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

The communication process ends when each processor holds all the messages it needs. Our communication model allows us to transmit each message at different times. This routing flexibility reduces the total communication time considerably. The instance given in Example 1.1 requires six communication steps if one restricts each message to be transmitted only at a single time unit. However, by allowing messages to be transmitted at different times one can perform all communications in four steps. This is best possible unless forwarding is allowed, in which case all communications can be carried out in three steps [6].

A *communication mode* C is a set of tuples (m, l, D), where l is a processor index ($1 \leq l \leq n$), and message $m \in h_l$ is to be multicasted from processor P_l to the set of processors with indices D. In addition all tuples in a communication mode C must obey the following communications rules imposed by our network:

1.- All the indices l in C are distinct, i.e., each processor sends at most one message; and

2.- Every pair of D sets in C are disjoint, i.e., every processor receives at most one message.

A *communication schedule* S for a problem instance I is a sequence of communication modes such that after performing all these communications 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 in schedule S, and our problem consists of constructing a communication schedule with least total communication time.

Multimessage multicasting problems arise when solving sparse systems of linear equations via iterative methods (e.g., a Jacobi-like procedure), most dynamic programming procedures, etc.

Previous Work, and New Results

The *basic multicasting problem* consists of all degree $d = 1$ MM_C problem instances, and can be trivially solved by sending all the messages at time zero. For pr-dynamic networks, communication modes can be performed in two stages: the data replication step followed by the data distribution step ([10], [14], [11]).

Gonzalez [5] 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. The interesting point is that each communication mode translates into a single communication step for processors interconnected via permutation networks.

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 ([1], [15], [2], [9], [8], [12]). Research has been limited to unicasting, and all known results about multicasting are limited to single messages, except for the work by Shen [13] who has studied multimessage multicasting for hypercubes. Since hypercubes are static networks, there is no direct comparison to our work.

Gonzalez [5] showed that even when $k = 2$ the MM_C problem is NP-hard. Gonzalez [3] developed an efficient algorithm to construct for any degree d problem instance a schedule with total communication time at most d^2, and presented problem instances for which this upper bound is best possible. Gonzalez [5], [3] and [4] developed several fast approximation algorithms for problems instances with any arbitrary degree d, but small fan-out ([3], [4]). These problem arise in practice, since the fan-out restriction is im-

posed by the applications and the number of processors available in commercial systems.

The MMF_C problem is also an NP-hard problem [5] and all the approximation results for the MM_C problem also solve the MMF_C problem.

In this paper we present an efficient algorithm 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 the previous approximation algorithms are faster than the one in this paper. However, our new algorithm generates communication schedules with significantly smaller total communication time. Our new algorithm 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 that needs to be solved is a MU_C problem of degree d. In the second phase we generate a schedule for this problem by reducing it to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time [7]. The solution is the concatenation of the schedules for each of these two phases.

2 Approximation Algorithm for the MMF_C Problem

We show that for every degree d instance of the MMF_C problem one can construct in polynomial time, with respect to the input length, a schedule with total communication time $2d$.

Problem instances will be represented by the tuple (I, G), where G is a directed multigraph (introduced in the previous section (see Figure 1)) that changes at each iteration, and $I = (P, H, N, d)$. We say that a processor is *critical* if it has d bundles emanating from it, otherwise it is called *noncritical*.

Our algorithm consists of four basic steps explained in the following subsections. As we describe our algorithm, we illustrate its operations by applying it to the problem instance given in figure 2 (solid objects).

2.0.1 Input Balancing

Given instance (I, G) of the MMF_C, we transform it to the balanced incoming message (all processors have equal number of incoming messages) problem $(\bar{I}, \bar{G})$ of the MMF_C as follows.

Problem $(\bar{I}, \bar{G})$ is (I, G), except that there are two additional processors, $\bar{P}_{n+1}$ and $\bar{P}_{n+2}$. Processor $\bar{P}_{n+1}$ sends d distinct messages to processor $\bar{P}_{n+2}$, and processor $\bar{P}_{n+2}$ sends d distinct messages to processor $\bar{P}_{n+1}$. In addition, the i^{th} message emanating from $\bar{P}_{n+2}$ is also transmitted to every processor with less than i total incoming messages in (I, G). All the new processors and messages are referred to as *dummy processors*, and *messages*.

Applying our transformation to the problem instance given in Figure 2 (solid objects) results in the addition of the dotted objects shown in Figure 2.

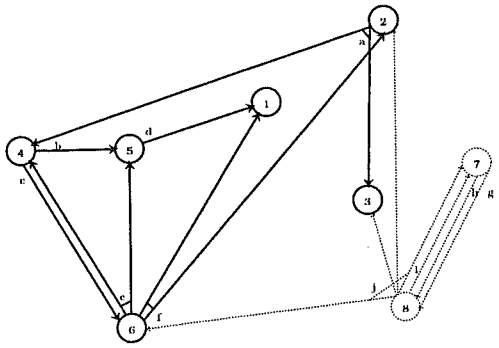

Figure 2: Solid objects represent (I, G), and all the objects represent $(\bar{I}, \bar{G})$.

Problem instance $(\bar{I}, \bar{G})$ is of degree $\bar{d} = d$, has $\bar{n} = n + 2$ processors, and each processor has exactly $\bar{d}$ incoming messages. Given any schedule for problem instance $(\bar{I}, \bar{G})$ one can easily construct a communication schedule for problem instance (I, G) by simply deleting all the dummy messages and dummy processors. This step takes $O(dn)$ time.

2.0.2 Transform to the MU_C Problem

This is the most involved step of our algorithm that transforms $(\bar{I}, \bar{G})$ to the instance $(\hat{I}, \hat{G})$ of the MU_C problem with $\hat{n} = \bar{n}$ processors and degree $\hat{d} = \bar{d}$. This transformation requires that a set of communications take place. We construct schedule X with total communication time $\hat{d} = \bar{d}$ for these message transmissions. The final schedule is X plus a schedule for $(\hat{I}, \hat{G})$ with total communication time equal to $\hat{d}$.

Our transformation begins by constructing (I_0, G_0) from $(\bar{I}, \bar{G})$. Then, for $1 \leq i \leq \bar{d}$, we construct (I_i, G_i) from (I_{i-1}, G_{i-1}). Problem instance (I_i, G_i), for $0 \leq i \leq \bar{d}$, is an instance of the MMF_C problem of degree $d_i = \bar{d}$ that satisfies the following two properties:

$W1$: Each processor in G_i has at least i single-edge bundles (unicasting messages) emanating from it.

$W2$: Every noncritical processor (processor needs to send less than d_i messages) in G_i contains only

single-edge bundles emanating from it, i.e., and its bundles represent unicasting messages.

By property $W1$, and the fact that $d_d = \bar{d}$, we know that (I_d, G_d) is also an instance of the MU_C problem and it will be referred to as $(\hat{I}, \hat{G})$.

Initially (I_0, G_0) is set to $(\bar{I}, \bar{G})$. Since $i = 0$, we know that (I_0, G_0) satisfies $W1$. Now we apply transformation T to (I_0, G_0) as many times as possible. *Transformation T:* For each noncritical processor with at least one multiple-edge bundle emanating from it, split one of its multiple-edge bundles into two bundles (a multiple-edge bundle and a single-edge bundle). Eventually all noncritical processors (if any) will consist only of single-edge bundles, i.e., it satisfies $W2$, and since the transformation does not decrease the number of single-edge bundles emanating from processors, it then follows that it also satisfies $W1$.

Applying transformation T to $(\bar{I}, \bar{G})$ given in Figure 2 we generate the problem instance (I_0, G_0) given in Figure 3 (all objects). The only difference (ignoring the different types of lines and circles) is that processor P_2 two single-edge bundles for message a instead of a two-edge bundle.

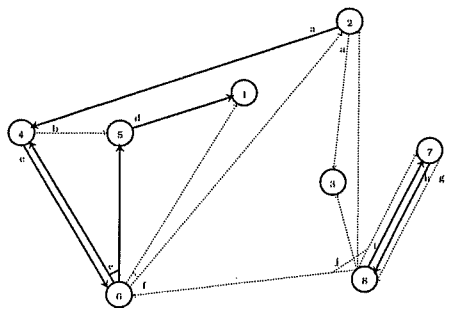

Figure 3: Final problem instance (I_0, G_0) (whole figure). The dotted lines correspond to the Q_js for $j \in S \bigcup S'$ when $i = 1$.

It is simple to prove that after $O(nd)$ steps the procedure terminates, and the resulting problem instance (I_0, G_0) satisfies properties $W1$ and $W2$.

Let us now show how to construct problem instance (I_i, G_i) of the MMF_C with degree $d_i = d_{i-1}$ that satisfies $W1$ and $W2$ from (I_{i-1}, G_{i-1}), for $1 \leq i \leq d$. Our procedure initializes (I_i, G_i) to (I_{i-1}, G_{i-1}). For each processor P_j we define Q_j to be a bundle with the largest number of edges emanating from it unless there are no bundles emanating from it in which case $Q_j = \emptyset$. All critical processors with exactly i single-edge bundles emanating from them are marked. Let S be the set of indices of the critical processors. Since (I_{i-1}, G_{i-1}) satisfies $W1$ and $W2$, we know that if one deletes from each processor in $j \in S \bigcup S'$, for some $S' \subseteq \{l \mid P_l$ is noncritical and P_l has at least i bundles emanating from it$\}$ the bundle Q_j, and if one can distribute all of this communication load by adding a single-edge bundle to the marked processors, and a single or multiple-edge bundle to the unmarked processors, then we end up with an equivalent instance that satisfies $W1$ and $W2$, provided that the set of messages that were redirected are transmitted to the appropriate processors. In what follows we specify this process in detail. Our procedure is more complex than needed because we want to use the same schedule for pr-dynamic networks without increasing the total communication time. This can be accomplished when each multicasting operation in the schedule sends each message to adjacent processors only.

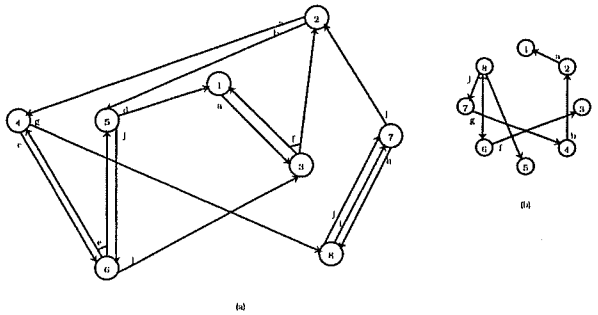

Figure 4: (a) Problem instance (I_1, G_1) before applying the transformation T. (b) Communication mode generated when $i = 1$.

$X \leftarrow \emptyset$;
for $i = 1$ **to** d **do**
 $(I_i, G_i) \leftarrow (I_{i-1}, G_{i-1})$;
 /* Modify (I_i, G_i) to satisfy $W2$*/
 /* The following references are for pair (I_i, G_i)*/
 Unmark all processors;
 $S \leftarrow \{j \mid$ processor P_j is critical$\}$;
 for each processor P_j **do**
 $Q_j \leftarrow$ set of edges (possibly empty) in a bundle with largest fan-out emanating from P_j;
 Mark P_j if it is critical and has exactly $i-1$ single-edge bundles emanating from it;
 endfor
 $S' \leftarrow \emptyset$;
 if $\sum_{j \in S} |Q_j| < n$ **then**
 /* We claim that the set S' exists. */
 Let $S' \subseteq \{1, 2, \ldots, n\}$ such that $\forall\, j \in S'$, P_j is noncritical and has at least i bundles emanating from it, and $\sum_{j \in S} |Q_j| + |S'| = n$;

/* Redistribution of Q_j for $j \in S \cup S'$ */
$l \leftarrow 0$;
for $j = 1$ **to** n **do**;
if $j \in S \bigcup S'$ **then** $bot \leftarrow l + 1$;
repeat
$l \leftarrow l + 1$;
case /* add to P_l */
:P_l is marked:
/* add a new single-edge bundle */
delete an edge from Q_j, and add it as a bundle emanating from P_l;
:else:
/* add a new single or multi-edge bundle */
$w =$ **min**$\{ |Q_j| , \sum_{j' \in S} |Q_{j'}| + |S'| - (n-l) \}$;
/* move the largest number of edges in Q_j, but leave enough for the remaining processors */
delete w edges from Q_j, and add them as a bundle emanating from P_l;
endcase
until $Q_j = \emptyset$;
$top \leftarrow l$;
During the i^{th} time period in communication schedule X multicast (unicast if $bot = top$) the message represented by bundle Q_j emanating out of P_j to processors $P_{bot}, P_{bot+1}, \ldots, P_{top}$;
Delete (empty) bundle Q_j emanating out of P_j;
Delete j from S or S';
endfor
Apply transformation T to (I_i, G_i) until $W2$ holds
endfor

The final problem instance (I_0, G_0) is given in figure 3 (all objects). The dotted lines correspond to the Q_js for $j \in S \bigcup S'$ when $i = 1$. Note that $S' = \emptyset$.

The new transmissions that appear in (I_1, G_1) (Figure 4 (a)) just before applying transformation T are as follows: Message a originates at P_1 rather than just at P_2, message b originates at P_2 rather than at P_4, message f originates at P_3 rather than at P_6, message g originates at P_4 rather than at P_7, and message j now originates at P_5, P_6, P_7, and P_8 rather than just at P_8. The communication mode generated when $i = 1$ is given in figure 4(b). This communication mode sends message a from P_2 to P_1 so that in (I_1, G_1) message a can be transmitted from P_1; sends message b from P_4 to P_2 so that in (I_1, G_1) message b can be transmitted from P_2; etc. Note that in this particular case there is no need to transmit message j from P_8 to P_7 because that has already been done in the communication mode. In what follows we will not comment again about this type of situations.

Transformation T generates the final problem instance (I_1, G_1) given in Figure 5 (all objects). The only difference from Figure 4 is that instead of message f emanating from P_3 as a two-edge bundle, it is now two single-edge bundles emanating from P_3. The dotted lines correspond to the Q_js for $j \in S \bigcup S'$ when $i = 2$. Note that $S' = \emptyset$.

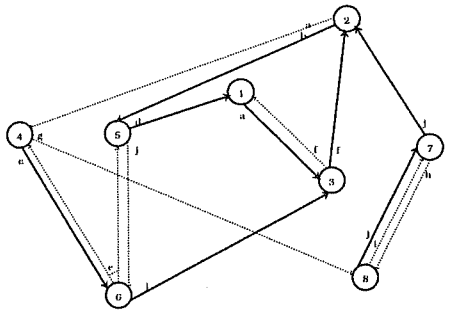

Figure 5: Final problem instance (I_1, G_1) (all objects). The dotted lines correspond to the Q_js for $j \in S \bigcup S'$ when $i = 2$.

The problem instance (I_2, G_2) and the communication mode generated when $i = 2$ is given in figure 6(b). We claim the following result without a proof.

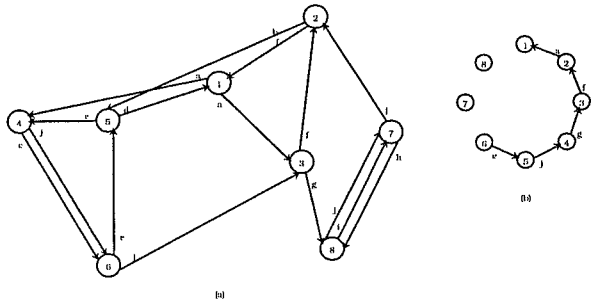

Figure 6: (a) Problem instance $(I_2, G_2) = (\hat{I}, \hat{G})$. (b) Communication mode generated when $i = 2$.

Lemma 2.1 *Problem $(I_d, G_d) = (\hat{I}, \hat{G})$ is an instance of the MU_C problem and schedule X plus any schedule for $(\hat{I}, \hat{G})$ is a schedule for $(\bar{I}, \bar{G})$. The time complexity for our procedure is $O(nd \log d)$*

2.0.3 Solving the MU_C Problem

We construct schedule (X') for the instance $(\hat{I}, \hat{G})$ of the MU_C problem of degree $\hat{d}$ with total communication time $\hat{d}$ by reducing it to the Makespan Openshop Preemptive Scheduling problem, which can be solved by the polynomial time algorithm given in [7]. The reduction appears in [5] and the time complexity is $O(r(min\{r, m^2\} + m \log m))$, where $r \leq \hat{d}\hat{n}$.

Problem instance $(\hat{I}, \hat{G})$ is given in Figure 7 (a) (all objects). The two communications modes generated for it are given in Figures 7 (b) and (c).

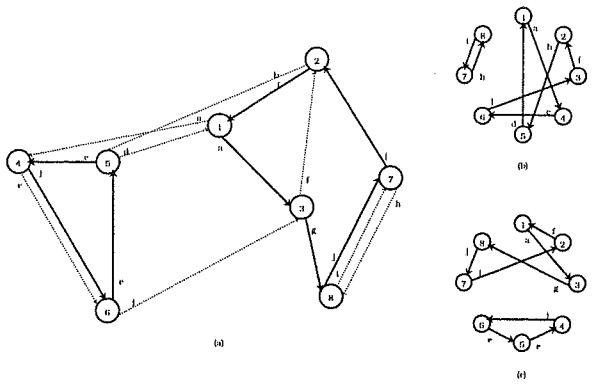

Figure 7: (a) Problem instance $(\hat{I}, \hat{G})$ (all objects). The communication modes (b) and (c) generated for the dotted lines, and solid lines in (a), respectively.

2.0.4 Constructing the Final Schedule

Concatenate the communication schedule X and X', and eliminate all dummy processors and messages. In our example thes schedules are given in Figures 4 (b), 6 (b), 7 (b) and 7 (c). The resulting communication schedule has total communication time at most $2d$ and it is the output generated by our procedure. We summarize our results in the following Theorem whose proof is omitted for brevity.

Theorem 2.1 *Schedule X plus schedule X' is a schedule for (I, G) with total communication time $2d$. The overall time complexity for our procedure is $O(r(min\{r, m^2\} + m\ log\ m))$, where $r \leq dn$.*

3 Discussion

The approximation algorithm in this paper generates a communication schedule with total communication time at most $2d$. This is significantly better than the one of previous algorithms ([3], [4]). However those algorithms are faster and were designed for the case when forwarding was not allowed. The time complexity for our procedure may be reduced by handling the dummy messages differently.

References

[1] E. G. Coffman, Jr, M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling File Transfers in Distributed Networks," *SIAM J. on Computing*, 14(3), pp. 744 – 780, 1985.

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

[3] T. F. Gonzalez, "Multi-Message Multicasting," Proceedings of Irregular'96, Lecture Notes in CS (1117), Springer, pp. 217-228, 1996.

[4] T. F. Gonzalez, "Improved Multimessage Multicasting Approximation Algorithms," Proceedings of PDCS'96, (1996), pp. 456 – 461, July 1996.

[5] T. F. Gonzalez, "Multi-Message Multicasting: Complexity and Approximations," Proceedings of HICSS-30, Jan. 1997.

[6] T. F. Gonzalez, "Multimessage Multicasting with Forwarding," UCSB, TRCS-96-24, Sep. 1996.

[7] T. F. Gonzalez, and S. Sahni, "Open Shop Scheduling to Minimize Finish Time," *JACM*, Vol. 23, No. 4, pp. 665 – 679, 1976.

[8] 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 Trans. on Comm.*, COM-30, 11, pp. 2475 – 2481, 1982.

[9] B. Hajek, and G. Sasaki, "Link Scheduling in Polynomial Time," *IEEE Trans. on Information Theory*, Vol. 34, No. 5, pp. 910 – 917, 1988.

[10] T. T. Lee, "Non-blocking Copy Networks for Multicast Packet Switching," *IEEE J. Selected Areas of Comm.*, Vol. 6, No 9, pp. 1455 – 1467, 1988.

[11] S. C. Liew, "A General Packet Replication Scheme for Multicasting in Interconnection Networks," *INFOCOM*, Vol.1 pp. 394 – 401, 1995.

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

[13] H. Shen, "Efficient Multiple Multicasting in Hypercubes," TR–95–20, School of Computing and Information Tech., Griffith University, Australia.

[14] J. S. Turner, "A Practical Version of Lee's Multicast Switch Architecture," *IEEE Trans. on Comm.*, Vol. 41, No 8, pp. 1166 – 1169, 1993.

[15] J. Whitehead, "The Complexity of File Transfer Scheduling with Forwarding," *SIAM Journal on Computing* Vol. 19, No 2, pp. 222 – 245, 1990.