Multimessage Multicasting with Forwarding

Teofilo F. Gonzalez
Department of Computer Science
University of California
Santa Barbara, CA, 93106
E-mail: teo@cs.ucsb.edu
Revised: October 16, 1996

Abstract

We consider Multimessage Multicasting over the n processor complete (or fully connected) static network (MM_C) with Forwarding. We present an efficient algorithm that constructs 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). Our algorithm consists of two phases. In the first phase a set of communications are scheduled to be carried out in d time periods in such a way that the resulting problem is a multimessage unicasting problem of degree d. In the second phase we generate a communication schedule for this problem by reducing it to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time. The final schedule is the concatenation of the communication schedules for these two phases.

Our centralized algorithms require all the communication information ahead of time. Applications where all of this information is readily available include iterative algorithms for solving linear equations, and most dynamic programming procedures. The Meiko CS-2 machine and in general computer systems with processors communicating via dynamic permutation networks whose basic switches can act as data replicators (e.g., n by n Benes network with 2 by 2 switches that can also act as data replicators) will also benefit from our results since the same schedules can be used in these systems.
Keywords: Approximation Algorithms, Multimessage Multicasting, Dynamic Networks, Parallel Iterative Methods, Communication Schedules, Forwarding.

1 Introduction

1.1 The Problem

The Multimessage Multicasting problem over the n processor static network (MM_C) consists of constructing a communication 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. In order to generate communication schedules with significantly smaller total communication time we allow *forwarding*, which means that a message can be sent to an intermediate processor or processors and then from some of these processors the message can be sent to other intermediate processors and/or to the processors that must receive the message. I.e., 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, and the 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. In many applications forwarding is allowed, but when security is an issue forwarding must not be permitted. For example technical reports may be sent indirectly, but it may be unacceptable to transmit credit card numbers, or other sensitive information indirectly when a direct link exists. Our introduction is a condensed version of the one in [8], which includes a complete justification for the multimessage multicasting problem as well as motivations, applications and examples.

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*. Multimessage Multicasting for pr-dynamic and complete networks is not too different, in the sense that any communication 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 pr-dynamic networks. The first phase replicates data and transmits it to other processors, and the second phase distributes data to the appropriate processors ([16], [17], [20]). The IBM GF11 machine [1], and the Meiko CS-2 machine use Benes networks for processor interconnection. The two stage translation process can also be used in the Meiko CS-2 computer system, and any multimessage multicasting schedule can be realized by using basic synchronization primitives. This two step translation process can be reduced to one step by increasing the number of network switches about 50% ([16], [17], and [20]). In what follows we concentrate on the MM_C problem because it has a simple structure, and, as mentioned above, results for this network can be easily translated to other dynamic networks.

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 discuss in Section 3 the MultiSource MMF_C in which initially messages may belong to several *hold* sets. 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. Consider the following example.

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. Note that in general processors may send and receive messages. The messages each processor holds and needs are given in Table 1. For this example the density d is* 3.

Table 1: Hold and Need vectors for Example 1.1.

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
$\{a,b\}$	$\{c,d\}$	$\{e,f\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
$\emptyset$	$\emptyset$	$\emptyset$	$\{a,c,e\}$	$\{a,d,f\}$	$\{b,c,e\}$	$\{b,d,f\}$	$\{c,d,e\}$	$\{c,d,f\}$

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.

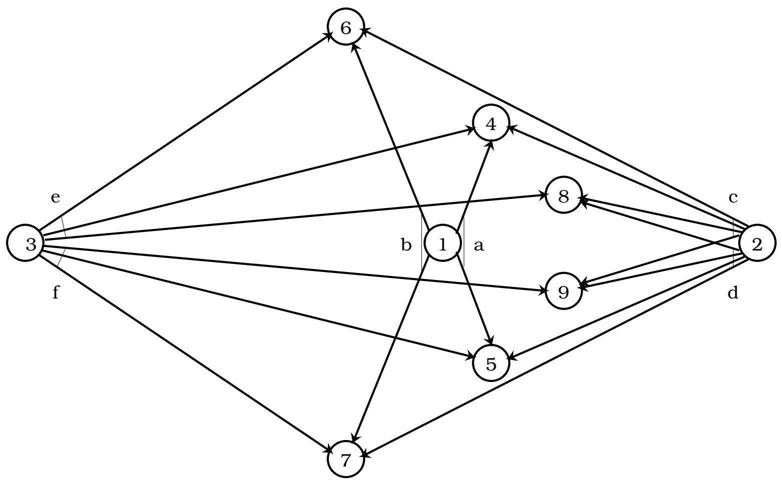

Figure 1: Directed Multigraph Representation for Example 1.1. The thin line joins all the edges (branches) in the same bundle

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

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

2.- During each time unit each processor may receive at most one message. The messages 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 has $n_i \subseteq h_i$, i.e., the processor holds all the messages it needs. Our communication model allows us to transmit any of the messages in one or more stages. I.e., any given message may be transmitted at different times. This added routing flexibility reduces the total communication time considerably. The problem instance given in Example 1.1 requires six communication steps if one restricts each message to be transmitted only at a single time unit. The reason for this is that no two of the six messages can be transmitted concurrently because every pair of messages either originate at the same processor, or have a common destination processor. However, by allowing messages to be transmitted at different times one can perform all communications in four steps. Let us now explain how this can be accomplished. In step $S1$ processor P_1 sends message a to processor P_5; processor P_2 sends message c to processors P_4, P_6, P_8 and P_9; and P_3 sends message f to processor P_7. In step $S2$ processor P_1 sends message a to processor P_4; processor P_2 sends message d to processors P_5, P_7, P_8 and P_9; and P_3 sends message e to processor P_6. In step $S3$ processor P_1 sends message b to processors P_6 and P_7; and processor P_3 sends message e to processor P_4 and P_8. In step $S4$ processor P_3 sends message f to processors P_5 and P_9. The last two messages where sent indirectly from their original location. Table 2 shows the hold vector at the end of each of these four steps.

Table 2: Hold vector after steps $S1$, $S2$, $S3$, and $S4$

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
$S1$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{c\}$	$\{a\}$	$\{c\}$	$\{f\}$	$\{c\}$	$\{c\}$
$S2$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, c\}$	$\{a, d\}$	$\{c, e\}$	$\{d, f\}$	$\{c, d\}$	$\{c, d\}$
$S3$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, c, e\}$	$\{a, d\}$	$\{b, c, e\}$	$\{b, d, f\}$	$\{c, d, e\}$	$\{c, d\}$
$S4$	$\{a, b\}$	$\{c, d\}$	$\{e, f\}$	$\{a, c, e\}$	$\{a, d, f\}$	$\{b, c, e\}$	$\{b, d, f\}$	$\{c, d, e\}$	$\{c, d, f\}$

To establish that forwarding reduces the total communication time we show that when forwarding is not allowed all the communication schedules for the problem instance given in Example 1.1 require at least four communication steps, but when forwarding is allowed, all the communications can be performed in three steps.

Let us now prove that without forwarding the problem instance given in Example 1.1 requires at least four communication steps. The proof is by contradiction. Suppose that all the communications can be carried in three steps. Then since there are only three communication steps and messages c and d originate from the same processor, then at least one of the messages $\{c, d\}$ must be transmitted to all its recipients at a single time unit. The same holds for messages a and b, and messages e and f. So assume without loss of generality that message c is transmitted to all its recipients at time t. At time t the only other messages that can be sent concurrently with message c are: message a or message f to processor P_5, and/or message b or message f to processor P_7. Since processors $P_4, P_5, \ldots, P_9$ must receive three messages, it must be that at each step each of these processors must receive one of the messages it needs. Therefore, at time t none of the processors may receive message e, and message f cannot be transmitted to all its recipients. This implies that message e must be sent to all its recipients at the same time. Let us say that this event occurs at time $t' \neq t$. But then processors P_4 and P_6 receive message c at time t, and message e at time t'. So in the remaining time unit processor P_6 must receive message b from P_1, and processor P_4 must receive message a from P_1. But, every processor (including P_1) may only send one message at a time. A contradiction. Therefore, without forwarding the problem instance given in Example 1.1 requires at least 4 time units to perform all its communications.

Let us now show that when forwarding is allowed all the communications in the problem instance given in Example 1.1 can be performed in three steps. In step $T1$ processor P_1 sends message a to processor P_4 and P_5; processor P_2 sends message c to processors P_6, P_8 and P_9; and P_3 sends message f to processor P_7. In step $T2$ processor P_1 sends message b to processors P_6 and P_7; processor P_2 sends message d to processors P_5, P_8 and P_9; and P_3 sends message e to processor P_4. In step $T3$ processor P_2 sends message c to processor P_4; processor P_3 sends message f to processors P_5 and P_9; processor P_4 sends message e to processors P_6 and P_8; and processor P_5 sends message d to processor P_7. Table 3 shows the hold vector at the end of each of these three steps.

Table 3: Hold vector after step $T1$, $T2$, and $T3$.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
$T1$	$\{a,b\}$	$\{c,d\}$	$\{e,f\}$	$\{a\}$	$\{a\}$	$\{c\}$	$\{f\}$	$\{c\}$	$\{c\}$
$T2$	$\{a,b\}$	$\{c,d\}$	$\{e,f\}$	$\{a,e\}$	$\{a,d\}$	$\{b,c\}$	$\{b,f\}$	$\{c,d\}$	$\{c,d\}$
$T3$	$\{a,b\}$	$\{c,d\}$	$\{e,f\}$	$\{a,c,e\}$	$\{a,d,f\}$	$\{b,c,e\}$	$\{b,d,f\}$	$\{c,d,e\}$	$\{c,d,f\}$

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

1.- All the indices o 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 applying all the communication modes $n_i \subseteq h_i$ for $1 \leq i \leq n$, i.e., every processor holds all the messages it needs. The *total communication time* is the latest time at which there is a communication which is equal to the number of communication modes in schedule S, and our problem consists of constructing a communication schedule with least total communication time. From the communication rules we know that every degree d problem instance has at least one processor that requires d time units to send, and/or receive all its messages. Therefore, d is a trivial lower bound for the total communication time. To simplify the analysis of our approximation algorithm we use this simple measure. Another reason is that load balancing (placement) and multimessage multicasting (routing) are normally separate procedures, and load balancing must use a simple objective function in terms of the problem instance it generates that somehow represents the final communication time for the particular placement and some reasonable routing procedure.

1.2 Previous Work, New Results, and Applications

The *basic multicasting problem* (BM_C) consists of all the degree $d = 1$ MM_C problem instances, and can be trivially solved by sending all the messages at time zero. There are no conflicts because $d = 1$, i.e., each processor must send at most one message and receive at most one message. The communication schedule has only one communication mode. When the processors are connected via a pr-dynamic network a communication mode can be performed in two stages: the data replication step followed by the data distribution step ([16],

[20], [17]). This two stage process can be used in the MEIKO CS-2 machine [8]. An important subproblem of the basic multicasting problem is when every message it to be sent to adjacent numbered processors. In this case this basic multicast operation can be performed in one step in pr-dynamic networks, and in the MEIKO CS-2 machine.

Gonzalez [8] 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 (e.g., Benes Network, Meiko CS-2, etc.), because in these networks all possible one-to-one communications can be performed in a single communication step.

It is not surprising that several authors have studied the MU_C problem as well as several interesting variations for which NP-completeness has been established, subproblems have been shown to be polynomially solvable, and approximation algorithms and heuristics have been developed. Coffman, Garey, Johnson and LaPaugh [2] studied a version the multimessage unicasting problem when messages have different lengths, each processor can send (receive) $\alpha(P_i) \geq 1$ ($\beta(P_i) \geq 1$) messages simultaneously, and messages are transmitted without interruption (non-preemptive mode). Whitehead [22] considered the case when messages can be sent indirectly. The preemptive version of these problems as well as other generalizations were studied by Choi and Hakimi ([4], [5], [3]), Hajek and Sasaki [14], Gopal, Bongiovanni, Bonuccelli, Tang, and Wong [12]. Some of these papers considered the case when the input and output units are interchangeable, i.e., each processor can be involved in at most $\gamma(P_i)$ message transmissions (sending and/or receiving). Rivera-Vega, Varadarajan and Navathe [18] studied, the file transferring problem, a version the multimessage unicasting problem for the complete network when every vertex can send (receive) as many messages as the number of outgoing (incoming) links. All previous work has been limited to unicasting, and all known results about multicasting are limited to single messages, except for the work by Shen [19] who has studied multimessage multicasting for hypercubes. This algorithms are heuristic and try to minimize the maximum number of hops, amount of traffic, and degree of message multiplexing. Since hypercubes are static networks, there is no direct comparison to our work. The MM_C problem involves multicasting of any number of messages, and its communication model is similar in nature to the one in the Meiko CS2 machine, after solving some basic synchronization problems.

The MM_C problem is significantly harder than the MU_C. Gonzalez [8] showed that even when $k = 2$ the decision version of the MM_C problem is NP-complete. Gonzalez [6] developed an efficient algorithm to construct for any degree d problem instance a communication schedule with total communication time at most d^2, and presented problem instances for which this upper bound on the communication time is best possible, i.e. the upper bound is also a lower bound. The lower bound holds when there is a huge number of processors and the fan-out is also huge. Since this situation is not likely to arise in the near future Gonzalez [6] and [7] considered the MM_C problem with restricted fan-out.

Gonzalez [8] also developed an algorithm to construct a communication schedule with total communication time $2d - 1$ for the case when the fan-out is two, i.e., $k = 2$. Gonzalez

[8] developed an $O(q \cdot d \cdot e)$ time algorithm, where $e \leq nd$ (the input size), to construct for problem instances of degree d a communication schedule with total communication time $qd + k^{\frac{1}{q}}(d-1)$, where q is the maximum number of colors that can be used to color each bundle and $k > q >= 2$.

Gonzalez [6] and [7] also developed several fast approximation algorithms with improved approximation bounds for problems instances with any arbitrary degree d, but small fan-out. These problem instances are the ones that arise in practice, since the fan-out restriction is imposed by the applications and the number of processors available in commercial systems. The approximation bound for these methods is about $(\sqrt{k}+1)d$, where k is the fan-out.

It is simple to show that the NP-completeness reduction for the MM_C ([8]) can be easily modified to establish the NP-completeness for the MMF_C problem. All the approximation results for the MM_C problem also hold for the MMF_C problem. However, for $d > 2$ it is impossible to prove that d^2 communication steps are required even for one instance of 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 presented in this paper. However, our new algorithm generates communication schedules with significantly smaller total communication time. Our 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 problem becomes a multimessage unicasting problem of degree d. In the second phase we generate a communication schedule for this problem by reducing it to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time. The solution is the concatenation of the communication schedules for each of these two phases.

Our multimessage multicasting problems arise when solving sparse systems of linear equations via iterative methods (e.g., a Jacobi-like procedure), and most dynamic programming procedures. Let us now discuss the application involving linear equations in more detail. We are given the vector $X(0)$ and we need to evaluate $X(t)$ for $t = 1, 2, \ldots$, using the iteration $x_i(t+1) = f_i(X(t))$. But since the system is sparse every f_i depends on very few terms. A placement procedure assigns each x_i to a processor where it will be computed at each iteration by evaluating $f_i()$. Good placement procedures assign a large number of $f_i()$s to the processor where the vector components it requires are being computed, and therefore can be computed locally. However, the remaining $f_i()$s need vector components computed by other processors. So at each iteration these components have to be multicasted (transmitted) to the set of processors that need them. The strategy is to compute $X(1)$ and perform the multimessage multicasting, then compute $X(2)$ and perform the multicasting, and so on. The same communication schedule is used at each iteration, and it can also be used to solve other systems with the same structure, but different coefficients. Speedups of n for n processor systems may be achieved when the processing and communication load is balanced, by overlapping the computation and communication time. This may be achieved by executing two concurrent tasks at each processor. One computes the x_is, beginning with the ones that need to be multicasted first, and the other deals with the multicasting of the x_i values. If all the transmissions can be carried out by the time the computation of all the x_is is finished then we have great performance. But if the communication takes too long compared to the computation, then one must try another placement or try alternative methods. What our algorithm will try to accomplish is to make the communication time as small as possible. In fact, it will be twice of optimal or $2d$.

2 Approximation Algorithm for the MMF_C Problem

In this section we show that for every degree d instance of the MMF_C problem one can construct in polynomial time, with respect to input length, a schedule with total communication time $2d$. The idea is to transmit a set of messages through a set of communication modes in such a way that the resulting problem is a multimessage unicasting problem, which can be solved in polynomial time. Before we present our algorithm, modify our previous notation and define additional terms.

One of the disadvantages of the notation used in the previous section for the MMF_C problem is that after the transmission of the messages in the first communication mode, several process will be holding the same message and it becomes difficult to decide which of these processors should be the one to transmit the message at subsequent steps. To minimize the time required to make this decision, our algorithm will at all times keep a list of messages that each processor will be transmitting to other processors. Because of this we also represent problem instances by the directed multigraphs introduced in the previous subsection (see Figure 1). Let $I = (P, H, N, n)$ be any degree d problem instance of the MMF_C with the specific communication requirements represented by multigraph G. In what follows we represent a problem instance by the tuple (I, G). We say that a processor is *critical* if it has d bundles emanating from it, otherwise it is called *noncritical.*

Our algorithm consists of the following four basic steps, which we define precisely and motivate in the subsections that follow.

1. **Transform to Uniform Incoming Message Problem Instance:** Transform problem instance (I, G) to the uniform incoming message problem instance $(\bar{I}, \bar{G})$ of the MMF_C with $\bar{n} = n + 2$ processors, degree $\bar{d} = d$, and all processors receiving exactly $\bar{d}$ distinct messages. The transformations adds two (dummy) processors and a set of (dummy) messages to accomplish the balancing. The reason for this step is to generate a uniform problem instance, and simplify subsequent steps of the algorithm and the correctness proof.

2. **Transform to Multimessage Unicasting Problem Instance:** Transform problem instance $(\bar{I}, \bar{G})$ to the instance $(\hat{I}, \hat{G})$ of the MU_C with $\hat{n} = \bar{n}$ processors and degree $\hat{d} = \bar{d}$. This transformation requires that a set of communications take place. We construct the communication schedule X with total communication time $\bar{d}$ for these message transmissions. The final communication schedule is X plus a communication schedule for $(\hat{I}, \hat{G})$.

3. **Constructing Communication Schedule for the Multimessage Unicasting Problem Instance:** Apply the reduction in Gonzalez [8] to problem instance $(\hat{I}, \hat{G})$ of the MU_C problem to generate the instance R of the openshop problem. Solve the openshop

problem R by using Gonzalez and Sahni's algorithm [11] and construct from it a communication schedule (X') for problem instance $(\hat{I}, \hat{G})$ with total communication time equal to $\hat{d}$.

4. **Constructing Final Communication Schedule:** Concatenate the communication schedules X and X', and eliminate all dummy processors and dummy messages. The resulting communication schedule has total communication time at most $2d$, and it is the output generated by our algorithm.

In the following subsections we show how to implement each of the above steps of our algorithm, and establish correctness for each of the steps. To get a better insight of our algorithm we will apply it to the problem instance given in figure 2 (solid lines). The problem instance consists of six processors, six messages, and has degree $d = 2$.

2.0.1 Transforming to Input Balanced Problem Instance

Given the degree d problem instance (I, G) of the MMF_C, we now show how to transform it to the uniform incoming message problem instance $(\bar{I}, \bar{G})$ of the MMF_C problem with $\bar{n} = n + 2$ processors, degree $\bar{d} = d$, and all processors receiving exactly $\bar{d}$ distinct messages. The balancing is accomplished by adding two (dummy) processors, and a set of (dummy) messages. The main reason behind this step is to generate a uniform problem instance, and simplify subsequent steps of the algorithm and its proof.

Problem instance $(\bar{I},\ \bar{G})$ is exactly like problem instance (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 problem instance (I, G). All the new processors and messages are called *dummy processors*, and *dummy messages*. The multigraph $\bar{G}$ is a copy of G plus two additional vertices (for the dummy processors), and appropriate branches and bundles for the dummy messages described above.

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.

It is simple to see that problem instance $(\bar{I},\ \bar{G})$ is of degree $\bar{d} = d$, has $\bar{n} = n + 2$, and each processor has exactly $\bar{d}$ incoming messages. Given any communication 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. It is simple to show that this step of our algorithm can be easily implemented to take $O(dn)$ time, which is proportional to the total number of edges in the multigraph. We formalize the above result in the following lemma.

Lemma 2.1 *Given any communication schedule for problem instance $(\bar{I},\ \bar{G})$ with degree $\bar{d} = d$, the procedure discussed above generates a communication schedule with identical communication time for problem instance (I, G) in $O(dn + cn)$ time, where dn is the total number of edges in the multigraph, and c is the total communication time of the schedule.*

Proof: By the above discussion.

$\square$

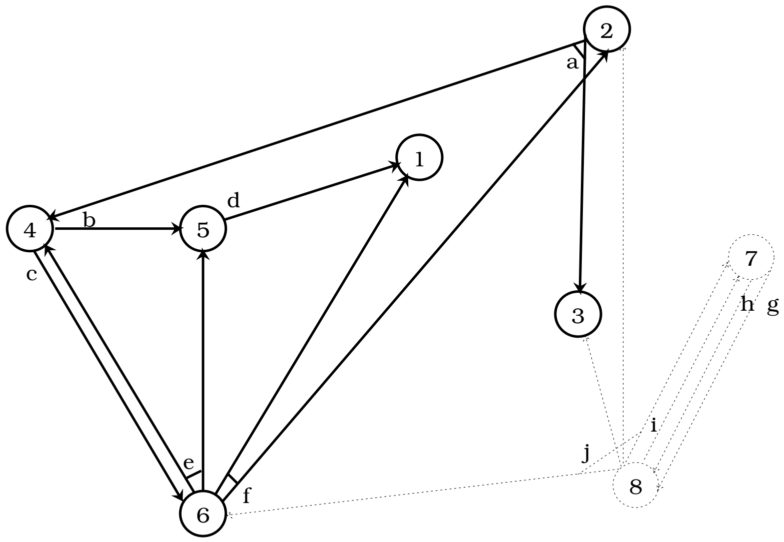

Figure 2: Solid objects represent (I, G) and all the objects represent $(\bar{I}, \bar{G})$.

2.0.2 Transform to Multimessage Unicasting Problem Instance

This is the most involved step of our algorithm. In this step we transform problem instance $(\bar{I}, \bar{G})$ 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 the communication schedule X with total communication time $\hat{d} = \bar{d}$ for these message transmissions. The final communication schedule is X plus a communication schedule for $(\hat{I}, \hat{G})$ with total communication time equal to $\hat{d}$.

Our transformation begins by constructing problem instance (I_0, G_0) from problem instance $(\bar{I}, \bar{G})$. Then for $1 \leq i \leq \hat{d}$ we construct problem instance (I_i, G_i) from problem instance (I_{i-1}, G_{i-1}). Problem instance (I_i, G_i) for $0 \leq i \leq \hat{d}$ is an instance of the MMF_C problem of degree $d_i = \hat{d}$ 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 in G_i contains only single-edge bundles emanating from it, i.e., bundles represent unicasting messages.

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

Lets begin by defining problem instance (I_0, G_0). 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 the following transformation to (I_0, G_0) as many times as possible. For each noncritical processor with at least one multiple-edge bundle emanating from it, increase its number of bundles by splitting 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 $W1$. The specific code to accomplish this is given below.

$(I_0, G_0) \leftarrow (\bar{I}, \bar{G})$ /* Now let us modify (I_0, G_0) so that it satisfies $W2$*/
/* In what follows all references are for (I_0, G_0)*/
while a noncritical processor P_j has a multiple-edge bundle emanating from it **do**
  Let R_j be the set of edges in a multiple-edge bundle emanating from P_j;
  while Processor P_j is noncritical and $|R_j| > 1$ **do**
    delete an edge from R_j and add it emanating from P_j as a single-edge bundle;
  endwhile
  if $|R_j| = \emptyset$ **then** delete bundle R_j from P_j;
endwhile

Applying the above transformation to the problem instance $(\bar{I}, \bar{G})$ given in Figure 2 we generate the problem instance (I_0, G_0) given in Figure 3 (all objects). The only difference between them is that processor P_2 has a two-edge bundle for message a in $(\bar{I}, \bar{G})$, but such message is split into two single-edge bundles in (I_0, G_0).

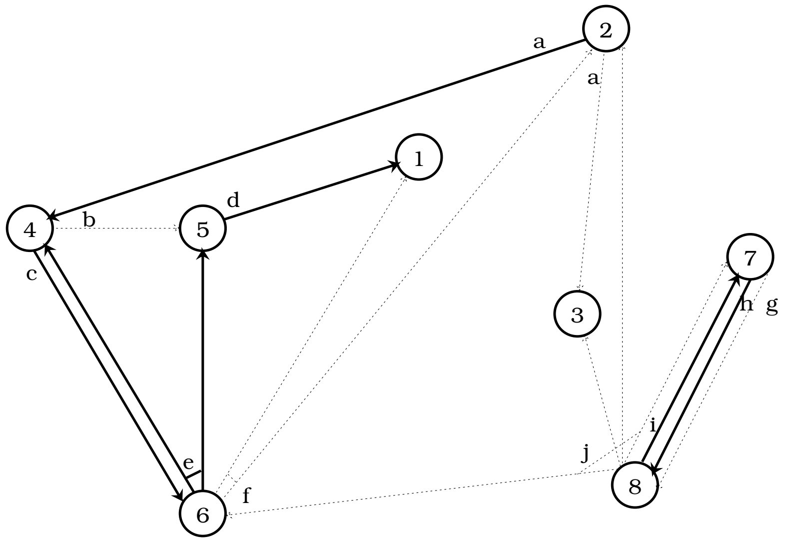

Figure 3: Final problem instance (I_0, G_0) (whole figure). The dotted lines correspond to the Q_js for $j \in S \bigcup S'$ when $i = 1$.

It is simple to prove that after no more than $O(nd)$ steps the procedure terminates, and at that point the resulting problem instance (I_0, G_0) satisfies properties $W1$ and $W2$.

Let us now show how to construct from (I_{i-1}, G_{i-1}), for $1 \leq i \leq d$, problem instance (I_i, G_i) of the MMF_C with degree $d_i = d_{i-1}$ that satisfies $W1$ and $W2$. Our procedure begins by initializing (I_i, G_i) to (I_{i-1}, G_{i-1}). For each processor P_j we let Q_j be a bundle with the largest number of edges emanating from it unless there are no bundles emanating from it in which case $Q_j = \emptyset$. We mark all critical processors that have exactly i single-edge bundles emanating from them. 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 marked processors, and a single-edge or a 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 more detail, give an example, and then we establish correctness. Our procedure is slightly more complex than needed because we want to use the same communication schedule for dynamic networks without increasing the total communication time. This can be accomplished when each multicasting operation in the communication schedule sends a message to adjacent processors only.

```
$X \leftarrow \emptyset$;
for $i = 1$ to $d$ do;
    $(I_i, G_i) \leftarrow (I_{i-1}, G_{i-1})$ /* Now let us modify $(I_i, G_i)$ so that it satisfies $W2$*/
    /* In what follows all references are for $(I_i, G_i)$*/
    Unmark all processors;
    for each processor $P_j$ do
        $Q_j \leftarrow$ set of edges in a bundle with largest fan-out emanating from $P_j$ unless,
                there are no bundles emanating from it in which case $Q_j = \emptyset$;
        $S = \{j \mid P_j$ is critical$\}$;
        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 following 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 $\sum_{j \in S} |Q_j| + |S'| = n$;
    /* Communication Redistribution */
    $l \leftarrow 0$;
    for $j = 1$ to $n$ do;
        if $j \in S \bigcup S'$ then
            $bot \leftarrow l + 1$;
            while $Q_j \neq \emptyset$ do;
                $l \leftarrow l + 1$;
                case
                    :$P_l$ is marked:
                        delete an edge from $Q_j$, and add it as a bundle emanating from $P_l$;
                    :else:
                        $w =$ min$\{|Q_j|, \sum_{j' \in S} |Q_{j'}| + |S'| - (n - l)\}$;
                        delete $w$ edges from $Q_j$, and add them as a bundle emanating from $P_l$;
                endcase
            endwhile
            $top \leftarrow l$;
            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) bundle $Q_j$ emanating out of $P_j$;
            Delete $j$ from $S$ if $j \in S$, otherwise delete $j$ from $S'$;
    endfor
    /* Transform $(I_i, G_i)$ so that it satisfies $W2$ */
    while a noncritical processor $P_j$ has a multiple-edge bundle emanating from it do
        Let $R_j$ be the set of edges in a multiple-edge bundle emanating from $P_j$;
        while processor $P_j$ is noncritical and $|R_j| > 1$ do
            delete an edge from $R_j$ and add it as a single-edge bundle emanating from $P_j$;
        endwhile
        if $|R_j| = \emptyset$ then delete bundle $R_j$ from $P_j$;
    endwhile
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 the transformation that makes it satisfy $W2$ are as follows: Message a originates at P_1 rather than 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 was done in the communication mode. Our algorithm could be modified to detect cases like this, but in general the total time complexity will not be reduced and the communication schedules will have the same total communication time. Also note that the algorithm sends message j to P_5, P_6, P_7 and P_8 (i.e., *bot* $= 5$ and *top* $= 8$), but there is no need to send it to P_8. Again, our algorithm does not detect these cases that when eliminated will decrease the total communication time in some cases, but we have not figured out how to apply these simplifications to decrease the total communication time, and/or decrease the time complexity bound all of the time. In what follows we will not comment again about this type of situations.

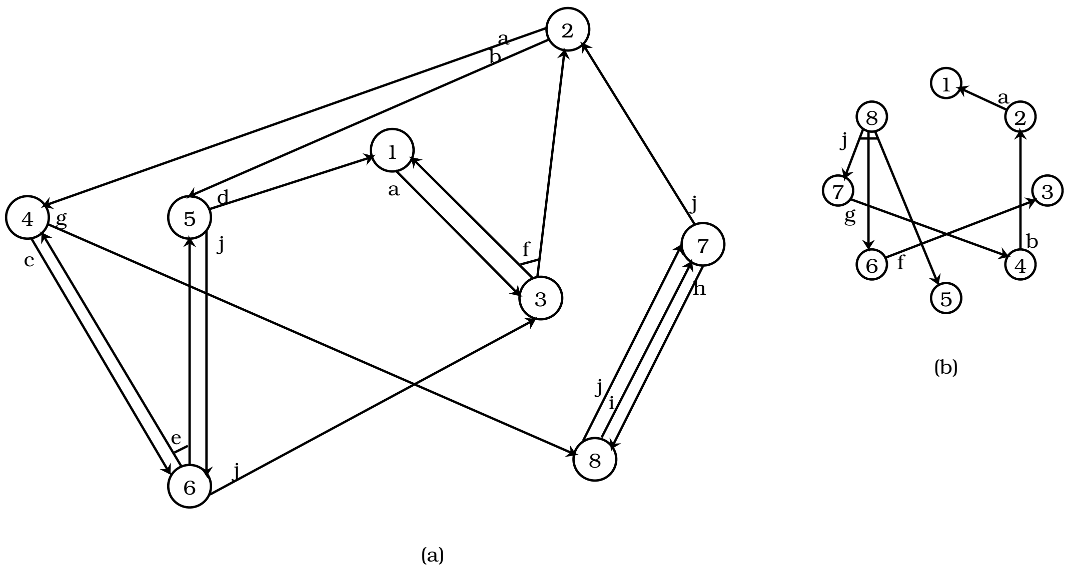

Figure 4: (a) Problem instance (I_1, G_1) before applying the transformation that makes it satisfy $W2$. (b) Communication mode generated when $i = 1$.

The transformation that makes (I_1, G_1) satisfy W_2 generates the final problem instance (I_1, G_1) given in Figure 5 (all objects). The only difference from Figure 2 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 $S' = \emptyset$.

The new messages that appear in (I_2, G_2) (Figure 6 (a)) just before applying the transformation that makes it satisfy $W2$ are as follows: Message a originates at P_1 rather than at P_2, message f originates at P_2 rather than at P_3, message g originates at P_3 rather than

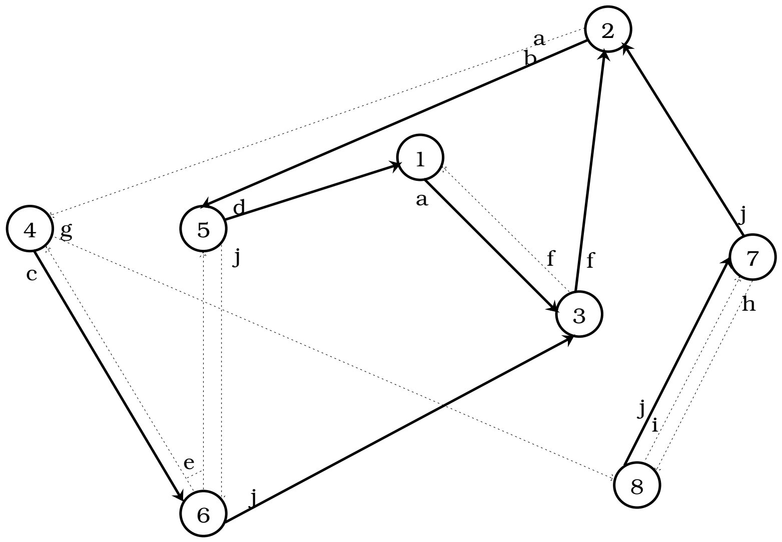

Figure 5: Final problem instance $(I_1,\, G_1)$ (all objects). The dotted lines correspond to the Q_js for $j \in S \bigcup S'$ when $i = 2$.

at P_4, message j originates at P_4 rather than at P_5, and message e originates at P_5 and P_6 rather than just at P_6. Note that the algorithm also makes the following changes that have no effects, message h originates at P_7 rather than at P_7, and message i originates at P_8 rather than at P_8. The communication mode generated when $i = 2$ is given in figure 6(b). In this communication mode, we send message a from P_2 to P_1 so that in $(I_1,\, G_1)$ message a can be transmitted from P_1; we send message f from P_3 to P_2 so that in $(I_1,\, G_1)$ message f can be transmitted from P_2; etc. There are no changes by the transformation that makes $(I_2,\, G_2)$ satisfy $W2$, because it already satisfies $W2$.

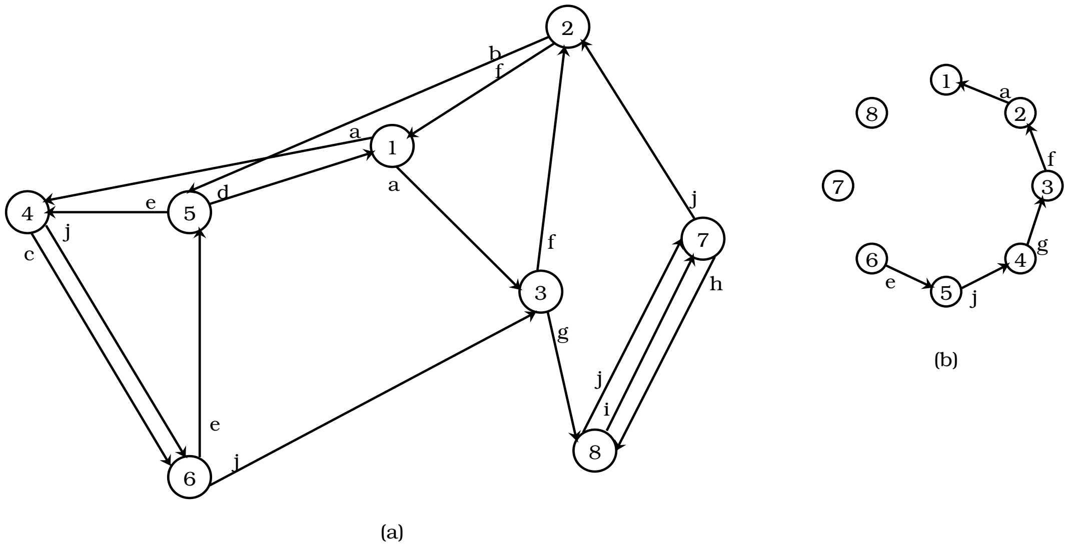

Figure 6: (a) Problem instance $(I_2,\, G_2) = (\hat{I},\, \hat{G})$. (b) Communication mode generated when $i = 2$.

Let us now prove the following lemma that we use later on to establish correctness.

Lemma 2.2 *Just after i is assigned a new value,*

$V1:$ *(I_{i-1}, G_{i-1}) is an instance of the MMF_C problem of degree $d_{i-1} = \bar{d}$ that satisfies properties $W1$ and $W2$; and*

$V2:$ *A communication schedule for $(\bar{I}, \bar{G})$ is the communication schedule X so far constructed plus a communication schedule for (I_{i-1}, G_{i-1}).*

Proof: First we show that (I_0, G_0) is an instance of the MMF_C problem of degree $d_0 = \bar{d}$ that satisfies properties $W1$ and $W2$. Initially (I_0, G_0) is just $(\bar{I}, \bar{G})$, and it obviously satisfies $W1$. Then, while there is a noncritical processor P_j with a multiple-edge bundle R_j emanating from it, we apply the following transformation. An edge is deleted from bundle R_j and it is reattached as a single-edge bundle emanating from P_j. Clearly, the number of bundles emanating from P_j is at most $d_0 = \bar{d}$, and the new problem instance is equivalent to the previous one in the sense that the same set of messages have to reach the same set of destinations. What has changed is that we have identified a message (R_j) that will be sent to one of its destinations at a different time than the rest of the destinations (but this could change later on). Therefore, when the transformation terminates (I_0, G_0) satisfies $W1$, and $W2$, and $V1$ holds when i receives the value 1.

Initially X is $\emptyset$, and (I_0, G_0) is equivalent to $(\bar{I}, \bar{G})$. Therefore, a communication schedule for $(\bar{I}, \bar{G})$ is any communication schedule for (I_0, G_0), and $V2$ holds when i receives the value 1.

We know show that if $V1$ and $V2$ hold when i receives some value ($[1..d]$), then properties $V1$ and $V2$ hold when i receives the next value.

Suppose properties $V1$ and $V2$ hold for some value for i ($[1..d]$), now we show that $V1$ and $V2$ hold for the next value for i. When i receives a value less than $d+1$, (I_i, G_i) is set to (I_{i-1}, G_{i-1}). All critical processors with exactly $i-1$ bundles in (I_i, G_i) are marked. For each processor P_j we let Q_j be a bundle with the largest number of edges emanating from it unless there are no bundles emanating from it in which case $Q_j = \emptyset$. Let S be the set of indices of the critical processors. Now there are two cases: (1) $\sum_{j \in S} Q_j \geq n$, or (2) there exists a subset $S' \subseteq \{1, 2, \ldots, n\}$ such that $\forall\ j \in S'$, P_j is noncritical and has at least i bundles emanating from it in (I_i, G_i), and $\sum_{j \in S} |Q_j| + |S'| = n$.

Let us now establish that if (1) does not hold, the (2) holds. The proof is by contradiction. Suppose that $\sum_{j \in S} |Q_j| + |S'| < n$, for the largest possible set S' such that $\forall\ j \in S'$, P_j is noncritical and has at least i bundles emanating from it in (I_i, G_i). Let q be the number of different values of j such that $j \notin S \bigcup S'$. Then the sum of the edges emanating from the processors P_j such that $j \notin S \bigcup S'$ is at most $q(i-1)$. The summation of the edges in the bundles emanating from the remaining processors is at most $(d-i+1)(n-1) + (i-1)(n-q)$ because $\sum_{j \in S} |Q_j| + |S'| < n$, the bundles included in Q_j are the largest in their processors, and all of these $n-q$ processors satisfy $W1$. So, the total number of edges emanating from the processors is at most $dn - d + i - 1$. Since $i \leq d$ it follows that the total number of edges emanating from processors is less than dn. But, all processors have exactly d incoming edges. A contradiction. Therefore, $\sum_{j \in S} |Q_j| + |S'| \geq n$.

Then the algorithm deletes one by one all of the Q_js such that $j \in S \bigcup S'$, where S' is defined by the algorithm, and attaches them to originate from a set of (adjacent) processors and ending at the same destinations. In order for this transformation to be correct, the schedule X is augmented with a communication mode that transmits the messages Q_j ($j \in S \bigcup S^{'}$) to the processors that will now be in charge of transmitting them. Therefore if we can show that by the end of the iteration (I_i, G_i) also satisfy $W1$ and $W2$, then $V1$ and $V2$ hold for the new value for i. Let us now show that that (I_i, G_i) satisfies $W1$ and $W2$.

The bundles that we delete are reattached to other processors as single-edge or multiple-edge bundles. The only processors where we can attach bundles with multiple edges are the unmarked ones, otherwise we might violate property $W1$. When $\sum_{j \in S} |Q_j| + |S^{'}| = n$ for the set $S^{'}$ we selected, then attaching them is simple since we just attach one destination for a message to each processor. When $\sum |Q_j| > n$ this process is more complex since we have to attach multiple edges to some processors. We now show that our algorithm will redistribute the communication load correctly.

For each processor with $Q_j \neq \emptyset$ such that $j \in S$ there is at least one processor (in some cases itself) that in not marked, and thus can be attached a multiple-edge bundle to it. So if $q = |S|$, then there are at most q processors that can be attached only one single-edge bundle, and there are at least q processors that may be attached multiple-edge bundles. So each of these q messages is reattached to zero or more unmarked processors plus some marked ones (if any), but at least to one processor. If we only attach each Q_j to zero or one unmarked processors then there are enough unmarked processors to absorb the load. On the other hand when we attach some Q_j to two or more unmarked processors then it must be that the total number of remaining edges that need to be redistributed is equal to the number of processors left in which case the remaining communication load can be reattached one message per processor. Therefore, all the bundles deleted is reattached to other processors.

To make the transformation valid one needs to multicast (unicast if $bot = top$) the message represented by Q_j to processors $P_{bot}, P_{bot+1}, \ldots P_{top}$, where bot and top are defined by the algorithm. All of the messages that need to be multicasted during the i^{th} iteration start and end at different processors, and the destination processors are adjacent. Therefore, it can be carried out in a single communication mode and it can be performed in a single communication step in dynamic networks.

At this point problem instance (I_i, G_i) satisfies $W1$, and now we apply the transformation that converts it into an instance that also satisfies W_2 (same one we applied to (I_0, G_0)). Therefore, $V1$ and $V2$ hold for the next value for i. This completes the proof of the lemma.

□

Corollary 2.1 *Problem instance $(I_d, G_d) = (\hat{I}, \hat{G})$ is also an instance of the MU_C problem and communication schedule X with total communication time $d_d = d$ constructed by the algorithm plus any communication schedule for $(\hat{I}, \hat{G})$ is a communication schedule for $(\bar{I}, \bar{G})$.*

Proof: Follows from the previous lemma.

□

Lemma 2.3 *The overall time complexity for our procedure that generates $(\hat{I}, \hat{G})$ from (I_d, G_d) is $O(nd \log d)$*

Proof: The overall time complexity for the procedure that makes the problem instances satisfy $W2$ is $O(dn)$ because the procedure is applied to noncritical processors at most d times, and once a processor becomes critical it will remain critical on the subsequent problem instances. The main loop is executed d times. The copy of one problem instance to the next does not need to be made, its existence is just to simplify the proof and notation. The unmarking and marking of processors takes $O(n)$. By storing the bundles in a heap for each processor on can find a bundle with largest number of edges in $O(log\ d)$ time. The other operations before the for loop j can easily shown to takes $O(n)$ time by using appropriate data structures. Since these structures are straight forward we do not elaborate on them. The loop for j can be shown to take $O(n)$ time because we visit each processor twice, one with j and the other with l, and each time it take constant amount of time. Therefore, each iteration takes $O(n\ log\ d)$ time, and the overall time complexity is $O(nd\ log\ d)$.

□

2.0.3 Constructing a Communication Schedule for the Multimessage Unicasting Problem Instance:

We now show that how to construct a communication schedule 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. This problem can be solved by the polynomial time algorithm given in [11].

An openshop consists of $m \geq 1$ machines, and $n \geq 1$ jobs. Each job consists of m tasks. The j^{th} task of job i $(T_{i,j})$ must be executed by the j^{th} machine for $t_{i,j} \geq 0$ time units. A schedule is an assignment of each task to its corresponding machine for a total of $t_{i,j}$ time units in such a way that at each time instance one task from each job may be assigned to a machine, and each machine may be assigned at most one task at a time. Note that the task processing need not be continuous, that is why this type of schedules are called *preemptive*. The finish time for schedule S $(f(S))$ is the latest time a task being processed by a machine. The makespan openshop scheduling problem consists of constructing a minimum finish time schedule.

Let m_i be the total time that machine i must be busy, and t_j be the total time that job j needs to be executed. Let $t = \ \mathbf{max}\{m_i, t_j\}$. Gonzalez and Sahni [11] have shown that there is always a preemptive schedule with finish time t, which is the best possible, and that one such schedule can be constructed in $O(r(min\{r, m^2\} + m\ log\ n))$ time, where r is the number of nonzero tasks. Furthermore, when all of the $t_{i,j}$s are integers, there is a schedule where preemptions occur only at integer points, and one such schedule is generated by Gonzalez and Sahni's [11] algorithm.

The MU_C problem of degree $\hat{d}$ is a special case of the preemptive openshop problem with every $t_{i,j} \in \{0, 1, \ldots, \hat{d}\}$. Each of the $\hat{n}$ vertices in the communication graph represents a job, and a machine. The multiset of edges T indicating that processor i must send $|T|$ messages to processor j is now translated to the statement that the j^{th} task of job i must be executed by machine j for $t_{i,j} = |T|$ time units. Translating the results from the openshop problem back to the communication problem, it means that every problem of degree $\hat{d}$, has a communication schedule with total communication time equal to $\hat{d}$ time units. Furthermore, one can easily adapt the algorithm for the minimum finish time openshop problem given in [11] to construct one such communication schedule. The time complexity is $O(r(min\{r, m^2\} + m\ log\ \hat{n}))$ time, where $r \leq \hat{d}\hat{n}$, and $m = \hat{n}$. This is formalized in the following lemma.

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).

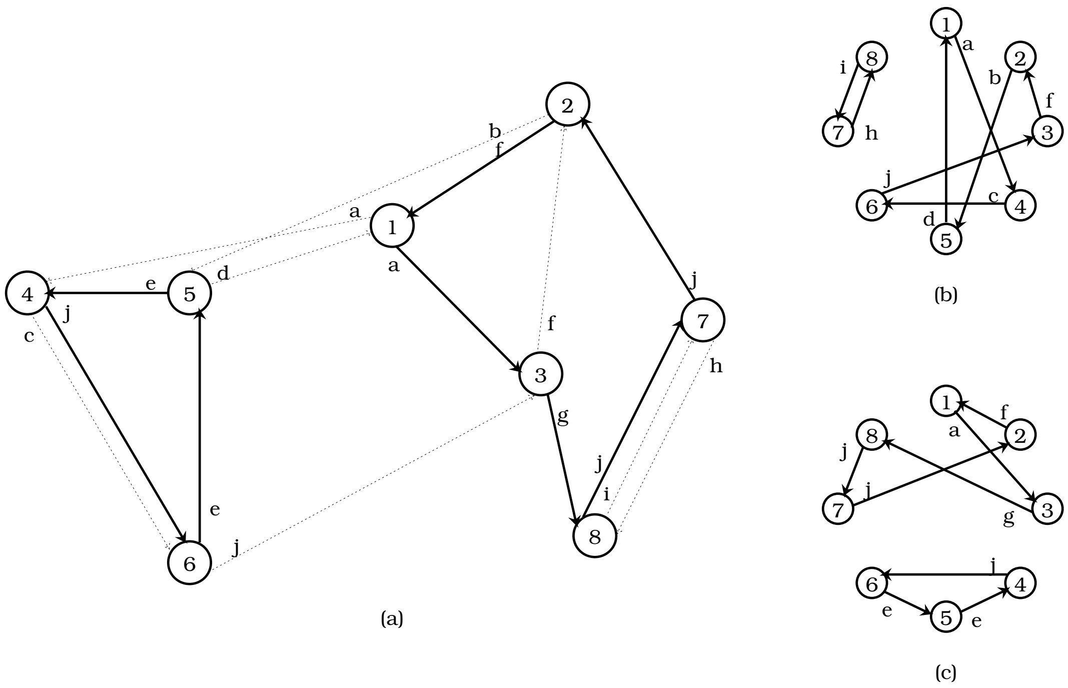

Figure 7: (a) Problem instance ($\hat{I}$, $\hat{G}$) (all objects). The communication modes (b) and (c) generated for the dotted lines, and solid lines in (a), respectively.

Lemma 2.4 *The above informal procedure constructs a communication schedule with total communication time equal to $\hat{d}$ for any multimessage unicasting problem $(\hat{I}, \hat{G})$ of degree $\hat{d}$ with $\hat{n}$ processors. The procedure takes $O(r(min\{r, m^2\} + m\ log\ \hat{n}))$ time, where $r \leq \hat{d}\hat{n}$, and $m = \hat{n}$.*

Proof: By the above discussion.

□

2.0.4 Constructing Final Communication Schedule:

Solve the openshop problem R by using Gonzalez and Sahni's algorithm [11] and construct from it a communication schedule (X') for problem instance $\hat{I}$ of the MU_C problem with total communication time equal to $\hat{d}$, as pointed out in the previous subsection.

Concatenate the communication schedule X with X' and eliminate all dummy processors and dummy messages. In our example, communication schedule X is given in Figures 4 (b) and 6 (b), and communication schedule X' is given in Figures 7 (b) and (c). At this point on can also eliminate all the messages originating at processor P_j and whose destination is the same processor P_j, since these communication is superfluous. The resulting communication schedule has total communication time at most $2d$ and it is the output generated by our procedure.

Theorem 2.1 *Communication schedule X produced by algorithm in Subsection 2.0.2 plus the communication schedule X' generated by the procedure in Subsection 2.0.3 is a communication schedule for (I, G) with total communication time $2d$. The overall time complexity for our procedure is $O(r(min\{r, m^2\} + m\ log\ n))$ time, where $r \leq dn$, and $m = n$.*

Proof: The proof follows from Lemmas 2.1, 2.3 and 2.4, and Corollary 2.1

□

3 MultiSource MMF_C

Let is now discuss our approximation algorithm for a generalization of the MMF_C problem that we refer to as the MultiSource MMF_C problem. The main difference is that initially messages may be present at several processors. Our algorithm reduces this problem to the MMF_C problem by selecting a unique origin for each message and ignoring the remaining places where it is located in such a way that we minimize the density of the resulting problem instance. Our algorithm is similar to a subalgorithm for an approximation algorithm for the MultiVia Assignment problem given by Gonzalez in [10].

The idea is to select an origin for each message in such a way that the resulting problem instance has least degree. Construct the following bipartite graph $G = (S \bigcup T, E)$, where S is the set of messages, T is the set of processors, and E is the set of edges defined as follows: there is an edge from vertex $s \in S$ to vertex $p \in P$ if message s is in processor p. In Figure 8 (a) shows five processors together with the set of messages each one holds at time zero. The bipartite graph G constructed from it is given in Figure 8 (b) (all edges).

An *s-matching* is G is a subset of edges not two of which are adjacent to the same vertex in S (for example $\{\{a, 4\}, \{b, 2\}, \{f, 4\}\}$ is an s-matching). A *complete s-matching* in G is an s-matching with cardinality $|S|$, i.e., each vertex in S has an edge in the complete s-matching associated with it. The set of dotted edges is a complete s-matching, but the set of solid edges in not a complete s-matching. For each complete s-matching I, we define $M(I)$ as the maximum number of edges in I incident to any node in T. We say that I is an *optimal complete s-matching* for G if it is a complete s-matching with least $M(I)$. The set of dotted edges is an optimal complete s-matching with $M(I) = 2$. It is simple to show that in this case there is more than one optimal complete s-matching. A polynomial time algorithm for finding an optimal complete s-matching is given by Gonzalez [10]. The algorithm finds a maximum matching in a set of bipartite graphs. The time complexity is $O(es^{1.5}t^{0.5}\ log$ s), where e is the number of edges, $s = |S|$, and $t = |T|$.

Our algorithm constructs the graph G and then finds an optimal complete s-matching I in it. Now the origin of each message that we have selected for each message is given by I as follows: each edge $(s, p) \in I$ indicates that message s will originate at processor p. The resulting problem is an instance of the MMF_C problem because every message originates at a single processor. Furthermore it has the least possible degree, because I is an optimal complete s-matching.

Theorem 3.1 *Our informal algorithm given above generates for any MultiSource MMF_C problem instance, and equivalent instance of the MMF_C problem with least possible degree. The time complexity for this algorithm is $O(em^{1.5}n^{0.5}\ log\ m)$, where e is the number of edges ($\leq nm$), n is the number processors, and m is the number of messages.*

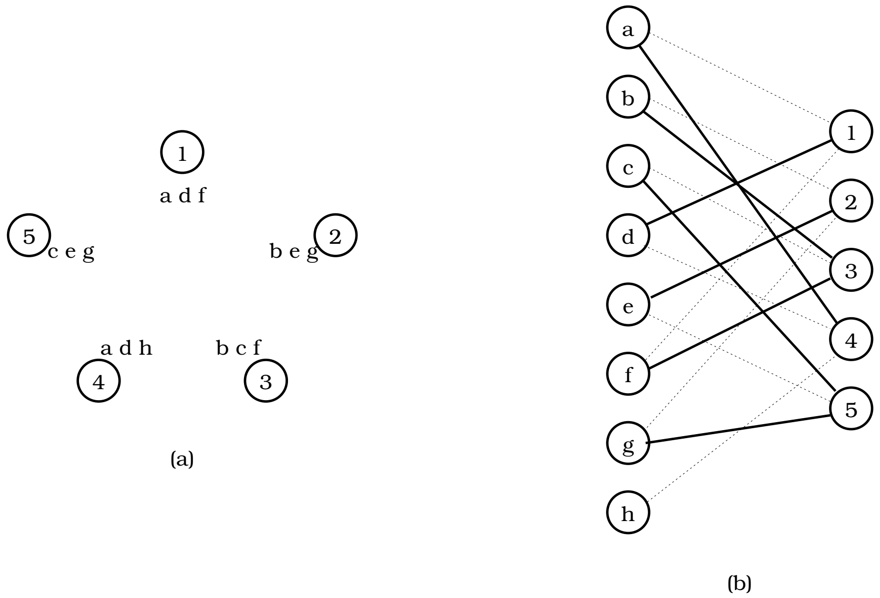

Figure 8: (a) MultiSource MMF_C problem instance. (b) Bipartite graph constructed from (a).

Proof: The proof follows from the above discussion.

□

4 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 ([6], [7]). However those algorithms are faster and were designed for the case when forwarding was not allowed.

The approach we have taken in this paper can be shown to require in the worst case $2d$ communication steps. The reason for this is when one processor contains d multi-edge bundles, then it requires d communication steps (modes) to transform the problem to a multimessage unicasting problem, and since the multimessage unicasting problem has degree d it requires also d communication steps (modes). On the other hand when each processor has at least l single-edge (unicasting) messages, and at most $d-l$ multi-edge (multicasting messages, then the total number of communication modes needed is just $2d-l$.

References

[1] G. S. Almasi, and A. Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Co., Inc., New York, 1994.

[2] 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) (1985), pp. 744 – 780.

[3] H.-A. Choi, and S. L. Hakimi, “Data Transfers in Networks,” *Algorithmica*, Vol. 3, (1988), pp. 223 – 245.

[4] H.-A. Choi, and S. L. Hakimi, “Scheduling File Transfers for Trees and Odd Cycles,” *SIAM Journal on Computing*, Vol. 16, No. 1, (1987), pp. 162 – 168.

[5] H.-A. Choi, and S. L. Hakimi, “Data Transfers in Networks with Transceivers,” *Networks*, Vol. 17, (1987), pp. 393 – 421.

[6] T. F. Gonzalez, “Multi-Message Multicasting,” Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems (Irregular’96), Lecture Notes in Computer Science (1117), Springer, (1996), pp. 217-228.

[7] T. F. Gonzalez, “Improved Multimessage Multicasting Approximation Algorithms,” Proceedings of the Ninth International Conference on Parallel and Distributed Computing Systems PDCS’96, (1996), (to appear).

[8] T. F. Gonzalez, “Multi-Message Multicasting: Complexity and Approximations,” Proceedings of the 30th Hawaii International Conference on System Sciences HICSS-30 (to appear). Also UCSB Department of Computer Science, Technical Report TRCS-96-15, July 1996.

[9] T. F. Gonzalez, “Improved Approximation Algorithms for Multimessage Multicasting,” UCSB Department of Computer Science, Technical Report TRCS-96-16, July 1996.

[10] T. F. Gonzalez, “An Approximation Algorithm for the Multi-Via Assignment Problem,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. CAD-3, No. 4, (1984), pp. 257 - 264.

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

[12] I. S. Gopal, G. Bongiovanni, M. A. Bonuccelli, D. T. Tang, and C. K. Wong, “An Optimal Switching Algorithm for Multibean Satellite Systems with Variable Bandwidth Beams,” *IEEE Transactions on Communications*, COM-30, 11 (1982) pp. 2475 – 2481.

[13] A J. Hopcroft, and R. M. Karp, “An $n^{2.5}$ Algorithm for Maximum Matchings in Bipartite Graphs,” *SIAM J. Computing,* (1973), pp. 225 – 231.

[14] B. Hajek, and G. Sasaki, “Link Scheduling in Polynomial Time,” *IEEE Transactions on Information Theory*, Vol. 34, No. 5, (1988), pp. 910 – 917.

[15] I. Holyer, “The NP-completeness of Edge-Coloring,” *SIAM J. Comp.,* 11 (1982), 117 – 129.

[16] T. T. Lee, “Non-blocking Copy Networks for Multicast Packet Switching,” *IEEE J. Selected Areas of Communication*, Vol. 6, No 9, (1988), pp. 1455 – 1467.

[17] S. C. Liew, “A General Packet Replication Scheme for Multicasting in Interconnection Networks,” *Proceedings IEEE INFOCOM ’95*, Vol.1 (1995), pp. 394 – 401.

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

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

[20] J. S. Turner, "A Practical Version of Lee's Multicast Switch Architecture," *IEEE Transactions on Communications*, Vol. 41, No 8, (1993), pp. 1166 – 1169.

[21] V. G. Vizing, "On an Estimate of the Chromatic Class of a p-graph," *Diskret. Analiz.*, 3 (1964), pp. 25 – 30 (In Russian).

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