Simple Algorithms for Multimessage Multicasting With Forwarding

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

December 19, 1997

Abstract

We consider Multimessage Multicasting over the n processor complete (or fully connected) static network (MM_C) when the Forwarding of messages is allowed. 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 each of these two phases. For $2 \leq l \leq d$ we present an algorithm to generate a communication schedule with total communication time at most $\lfloor (2 - \frac{1}{l})d \rfloor + 1$, for problem instances where each processor needs to send messages to at most ld destinations.

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 schedules generated by our algorithm 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 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 a single link direct 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 ([17], [18], [21]). 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% ([17], [18], and [21]). 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 pr-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* [1]*. 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.

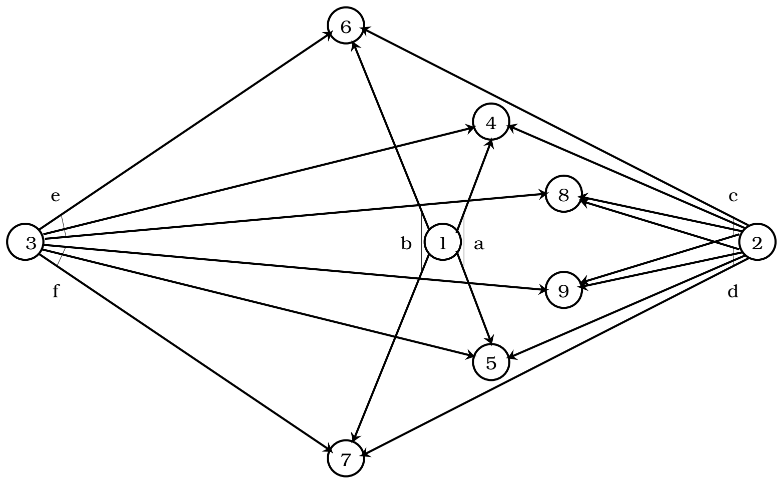

Figure 1: Directed Multigraph Representation for Example 1.1. The thin line joins all the edges (branches) in the same bundle.

The communications allowed in our complete network must 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.

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.

[1]Note that in general processors may send and receive messages.

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. In many cases it is a considerably reduction. Let us now give a problem instance such that the total communication time can be reduced by allowing this type of message routing. 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. 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. Since there are only three communication steps, and messages c and d originate at the same processor, then at least one of the messages $\{c, d\}$ must be transmitted to all its recipients during a single time unit. The same holds for messages $\{a, b\}$, and messages $\{e, f\}$. So assume without loss of generality that at time t message c is transmitted to all its recipients. The only other messages that can be sent at time t 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 together with the fact that messages e and f originate at the same processor 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, when forwarding is not allowed 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. The last two messages were sent indirectly from their original location. 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, 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 in 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 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 of these communications $n_i \subseteq h_i$ 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 sends at most one message and receives 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 ([17], [21], [18]). 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 ([17]), 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 [23] considered the case when messages can be sent indirectly. The preemptive version of these problems as well as other generalizations were studied by Choi and Hakimi ([4], [5], [3]), Hajek and Sasaki [15], Gopal, Bongiovanni, Bonuccelli, Tang, and Wong [13]. 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 [19] 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 the previous work has been limited to unicasting, and all known results about multicasting are limited to single messages, except for the work by Shen [20] who has studied multimessage multicasting for hypercube connected processors. His 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 considered the MM_C problem with restricted fan-out ([6], [7]).

Gonzalez [8] 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 also 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 $k > q \geq 2$ [8].

Gonzalez [6] and [7] developed several fast approximation algorithms with improved approximation bounds for problems instances with any arbitrary degree d, but small fan-out ([6], [7]). 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 problem given in [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 there exists at least one problem instance of the MMF_C problem that requires d^2 communication steps.

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). This algorithm is a simplified version of Gonzalez' algorithm [9]. We should point out that the previous approximation algorithms ([6], [7]) are faster than the one presented 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 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. For $2 \leq l \leq d$ we define the $l-MMF_C$ as the MMF_C in which each processor has at most ld edges emanating from it. We also present an algorithm to generate a communication schedule with total communication time at most $\lfloor(2-\frac{1}{l})d\rfloor+1$ for the $l-MMF_C$ problem.

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 in a parallel computing environment. 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 when they become available. If all the transmissions can be carried out by the time the computation of all the x_is is finished then we have achieved optimal performance. But if the communication takes too long compared to the computation, then one must try another placement or, if that fails one must try an alternate method. Our algorithm generates a schedule with total communication time at most twice of optimal (or $2d$) for any given placement.

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 the 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. The multimessage unicasting problem can be solved in polynomial time [12]. Before we present our algorithm we define additional terms.

One of the disadvantages of the notation given in the previous section for the MMF_C problem is that after the transmission of the messages in the first communication mode, several processors 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 the messages that each processor will be transmitting to other processors at a later time. This information is represented by a directed multigraph (introduced in the previous section (see Figure 1)) G that changes after each communication mode. We will also use the notation $I = (P, H, N, d)$, and a problem instance will be represented by the tuple (I, G).

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

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

2. **Construct a Communication Schedule for the Multimessage Unicasting Problem Instance:** Apply the reduction in [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 [12] and construct from it a communication schedule (X') for problem instance $(\hat{I}, \hat{G})$ with total communication time equal to $\hat{d}$.

3. **Construct the Final Communication Schedule:** Concatenate the communication schedules X and X'. The resulting communication schedule has total communication time at most $2d$, and is the output generated by our algorithm.

In the following subsections we show how to implement each of the above three steps of our algorithm, and then establish correctness. As we describe our algorithm, we illustrate its operations by applying it to the problem instance given in figure 2. The problem instance consists of 12 processors, 11 messages, and is of degree $d = 2$.

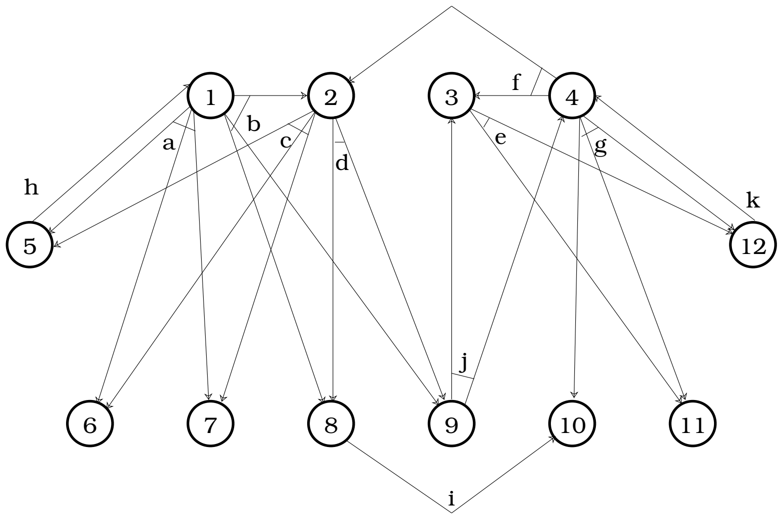

Figure 2: MM_C problem instance (I, G).

2.0.1 Transform to the Multimessage Unicasting Problem

This is the most involved step of our algorithm, where we transform problem instance (I, G) to the instance $(\hat{I}, \hat{G})$ of the MU_C problem with $\hat{n} = n$ processors and degree $\hat{d} = d$. This transformation requires that a set of communications take place. We construct the communication schedule X with total communication time $\hat{d}$ 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 $\hat{d}$. Procedure FORWARD, given below, constructs the instance $(\hat{I}, \hat{G})$ from (I, G). The idea is to forward all messages so that each processor ends up with at most d unicasting messages it needs to transmit. The forwarding is such that at each time unit each processor sends at most one message and receives at most one message, i.e., it obeys our communication rules. Our algorithm generates the specific operations to accomplish this based on two labelings and two functions (one indicates the message transmission time, and the other the message destination).

Procedure FORWARD (I, G)

Label B_i the i^{th} bundle visited while traversing the bundles emanating from P_1, then the ones emanating from P_2, and so on;

Define the function $t(i)$ as $(i-1) \bmod (d-1)+1$;

/* The message associated with bundle B_i will be forwarded at time $t(i)$. */

Label e_i the i^{th} edge visited while traversing the edges in B_1, then the ones in B_2, and so;

Define the function $g(i)$ as $\lceil \frac{i}{d} \rceil$;

/* Edge e_i will be forwarded to processor $P_{g(i)}$ */

Let $(\hat{I}, \hat{G}) \leftarrow (I, G)$ except that $\hat{G}$ does not have edges;

/* The edges in $\hat{G}$ will be added to reflect the forwarding operation. */

for every processor P_j in G **do**

  for every bundle B_i emanating out of P_j in G **do**

    Let $S = \{g(l) | e_l \in B_i\}$;

    Schedule X at time $t(i)$ will multicast the message associated with bundle B_i from processor P_j to the set of processors S (if $|S| = 1$, the operation is unicasting);

    for every edge $e_l \in B_i$ **do**

      Add to the hold set of processor $g(l)$ (i.e., $H(g(l))$, in $\hat{G}$ message B_i;

      Add the edge from $P_{g(l)}$ to P_q in $\hat{G}$, where q is the processor where edge e_l ends in G;

    endfor

  endfor

endfor

end of Procedure

In Figure 3 we show all the labelings performed by procedure FORWARD. The top number just below each of the vertices is the index of B_i, the next line shows the message name, and then by the value $t(i)$. The line just above the bottom one has the index of the e_j, and the bottom number is the processor index to where that edge is to be forwarded.

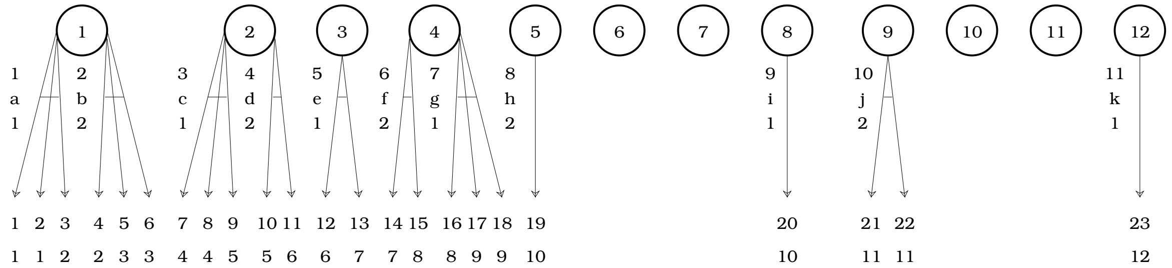

Figure 3: Labeling performed by procedure FORWARD.

All the communications in schedule X at time one and two are given in Figure 4. These communications are as follows. At time 1 message a is multicasted from processor P_1 to processors P_1 and P_2. Obviously one does not actually need to send the message to processor P_1, since P_1 holds that message. Our algorithm could be modified to detect cases like this one, but in general the total time complexity will not be reduced and the communication schedules will have the same total communication time. In what follows we will not comment again on this type of situations. At time 1 message c is multicasted from processor P_2 to processors P_4 and P_5; message e is multicasted from processor P_3 to processors P_6 and P_7;

message g is multicasted from processor P_4 to processors P_8 and P_9; message i is unicasted from processor P_8 to processor P_{10}; and message k is unicasted from processor P_{12} to processor P_{12} (superfluous operation). The communication operations for time 2 in schedule X are given in the forest labeled $T2$ in Figure 4.

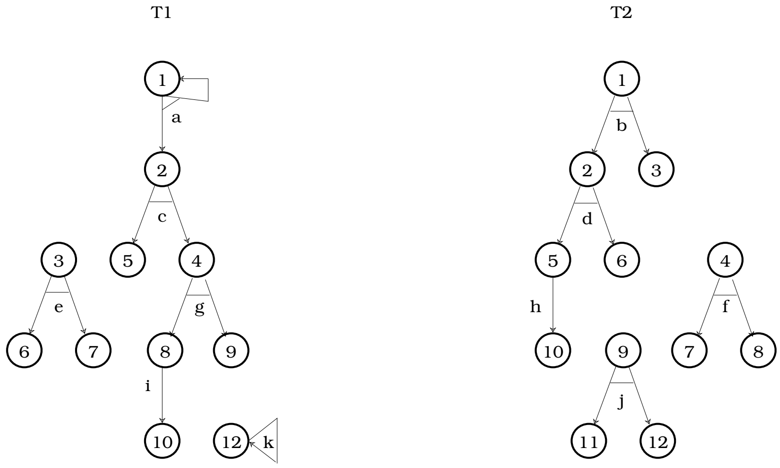

Figure 4: Schedule X at time one ($T1$) and time two ($T2$).

The resulting unicasting problem ($\hat{I}$, $\hat{G}$) of degree $\hat{d}$ is given in Figure 5 (all objects). Since the leftmost two edges in the bundle B_1 were forwarded to processor P_1 (superfluous operation), then message a is to be sent from processor P_1 to processor P_5 and P_6 in ($\hat{I}$, $\hat{G}$); the rightmost edge in bundle B_1 was forwarded to processor P_2, therefore message a needs to be sent to processor P_7 from P_2; the leftmost edge in bundle B_2 was forwarded to processor P_2, therefore message b needs to be sent to processor P_8 from P_2; the rightmost two edges in bundle B_2 were forwarded to processor P_3, therefore message b needs to be sent to processors P_9 and P_2 from P_3; the leftmost two edges in bundle B_3 were forwarded to processor P_4, therefore message c needs to be sent to processors P_5 and P_6 from P_4; the rightmost edge in bundle B_3 was forwarded to processor P_5, therefore message c needs to be sent to processor P_7 form P_5; the two edges in bundle B_4 were forwarded to processors P_5 and P_6, therefore message d needs to be sent to processors P_8 and P_9 from P_6; and so on. The resulting unicasting problem ($\hat{I}$, $\hat{G}$) of degree d is given in Figure 5.

Theorem 2.1 *Problem instance $(\hat{I}, \hat{G})$ an instance of the MU_C problem and communication schedule X with total communication time d constructed by procedure FORWARD plus any communication schedule for $(\hat{I}, \hat{G})$ is a communication schedule for (I, G).*

Proof: First we show that schedule X is a feasible schedule. From the $t()$ labels and procedure FORWARD, we know that for each processor the messages (if any) to be forwarded to other processors are multicasted at different times. From the $g()$ labels and procedure FORWARD we know that each processor will receive messages from at most d bundles and from the $t()$ labelings we know the messages associated with these bundles are transmitted at different times, thus all the messages forwarded to each processor are received at different times. From the function $t()$ and $g()$, and procedure FORWARD, we know that all the forwarding operation take placed during the d time periods. Therefore, X is a feasible communication schedule with total communication time d.

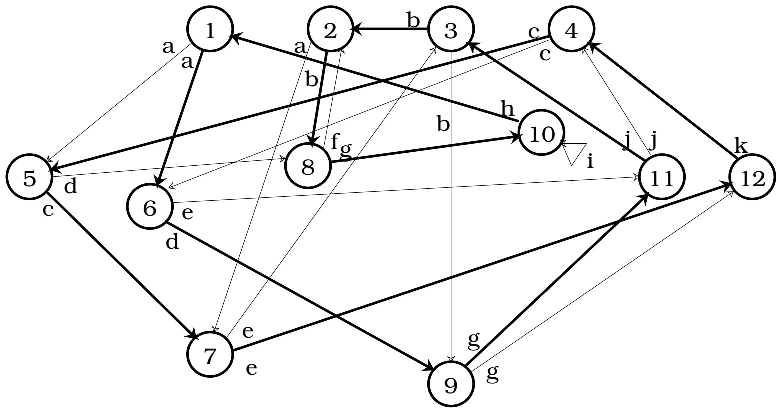

Figure 5: MU_C problem instance $(\hat{I}, \hat{G})$ constructed from (I, G) in Figure 2.

From the function $g()$ and procedure FORWARD we note that each message is forwarded to the appropriate processor so that if we carry out all the communications given by the resulting problem instance $(\hat{I}, \hat{G})$, we also solve problem instance (I, G). Therefore X plus any communication schedule for $(\hat{I}, \hat{G})$ is a communication schedule for (I, G).

□

Lemma 2.1 *The time complexity for procedure FORWARD is $O(n + e)$, where e is the total number of edges in (I, G).*

Proof: The steps before the first loop take $O(n + e)$. Overall, the innermost loop is executed once for each edge, the middle loop is executed once for each bundle, and the outermost loop is executed once for each processor. Therefore, the total time complexity is $O(n + e)$.

□

2.0.2 Construct a Communication Schedule for the Multimessage Unicasting Problem Instance:

We 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, which can be solved by the polynomial time algorithm given in [12]. This reduction appears in [8], but since it is a simple one we include it here for completeness.

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, denoted by $f(S)$, 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 [12] 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 [12] 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 [12] to construct one such communication schedule. The time complexity is $O(r(min\{r, \hat{n}^2\} + \hat{n} \ log \ \hat{n}))$, where $r \leq \hat{d}\hat{n}$. This is formalized in the following lemma.

Lemma 2.2 *([8]) 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 $O(r(min\{r, \hat{n}^2\} + \hat{n} \ log \ \hat{n}))$ time, where $r \leq \hat{d}\hat{n}$.*

Proof: By the above discussion.

$\square$

Problem instance $(\hat{I}, \hat{G})$ is given in Figure 5. The two communications modes generated for it are given in Figures 6 ($T3$) and ($T4$). Note that sending message b from processor P_3 to P_2 is not actually needed because processor P_2 already has it. This is a superfluous operation which can be deleted.

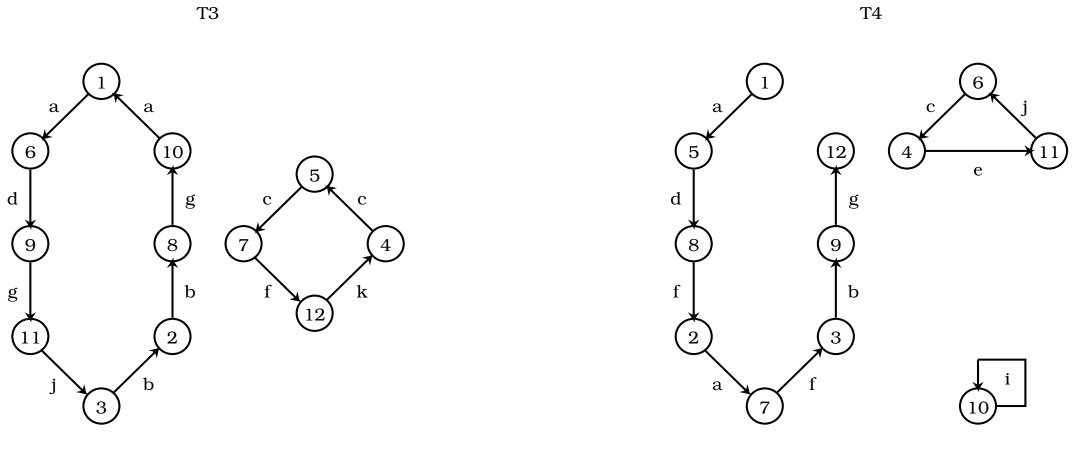

Figure 6: Two communication modes for problem instance $(\hat{I}, \hat{G})$.

2.0.3 Construct the Final Communication Schedule

Solve the openshop problem R by using Gonzalez and Sahni's algorithm [12] 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'. In our example, communication schedule X is given in Figures 4 (T1) and (T2), and communication schedule X' is given in Figures 6 (T3) and (T4). The resulting communication schedule has total communication time at most $2d$ and it is the output generated by our procedure.

Theorem 2.2 *Communication schedule X generated by procedure FORWARD 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 $O(r(min\{r, n^2\} + n \; log \; n))$, where $r \leq dn$.*

Proof: The proof follows from Lemmas 2.1, and 2.3, and Theorem 2.1 together with the fact that $n = \hat{n}$ and $d = \hat{d}$.

□

It is important to note that all the multicasting operations are to adjacent numbered processors. This is important since each communication mode composed of only those type of multicasting operations can be translated to a single communication mode for pr-dynamic networks. This implies that our the same schedule can be used for pr-dynamic networks.

Theorem 2.3 *Communication schedule X plus X' is a communication schedule with total communication time $2d$ for any pr-dynamic network.*

Proof: By the above discussion.

□

In Section 5 we discuss ways to decrease the total number of messages that need to be transmitted following the ideas behind the algorithm given in [9].

3 Approximation Algorithm for the $l - MMF_C$ problem

We present in this section our algorithm to generate a communication schedule with total communication time at most $\lfloor (2 - \frac{1}{l})d \rfloor + 1$ for the $l - MMF_C$ problem, for $2 \leq l \leq d$. Our procedure is identical to the one in the previous section, except that procedure FORWARD is replaced by l-FORWARD. The main difference between these two procedures is that only those processors that initially had more than d edges may forward messages, and a subset of processors, including those with at most $d + 1$ outgoing edges, will receive messages to be forwarded. After this forwarding operation we have reduced our problem to a multimessage unicasting problem which is reduced to the openshop problem and then solved as in the previous section. The openshop problem has degree d, and thus one can construct for it a schedule with total communication time at most d. The main difference in the final schedule is that the forwarding portion has a total communication time at most $d - \lfloor \frac{d}{l} \rfloor + 1$, and therefore the resulting schedule has total communication time at most $\lfloor (2 - \frac{1}{l})d \rfloor + 1$. It is important to point out we do not allow processors to forward only one edge, because the forwarding would be impossible within the above time bound when we have d of these processors and only one processor receiving all the forwarded messages.

Procedure l**-FORWARD** (I, G) /* Remember that $2 \leq l \leq d$ */
  for i=1 **to** n **do**
    while P_i has fewer than d bundles and at least one multi-edge bundle **do**
      delete an edge from a multi-edge bundle and add it as a single-edge bundle;
    endwhile
  endfor
  Let G_i be the number of edges emanating out of P_i in G;
  The tentative number of edges to be forwarded from P_i is

$$F_i = \begin{cases} 0 & \text{if } G_i \leq d \\ 2 & \text{if } G_i = d+1 \\ G_i - d & \text{if } G_i \geq d+2 \end{cases}$$

  Traverse the bundles emanating out of P_i in nondecreasing order with respect to the number of edges in the bundle, and visit all the edges in each bundle in any order.
  Mark the first F_i edges emanating out of P_i visited during the above traversal, and also mark all the bundles emanating out of P_i with at least one marked edge.
  Now traverse all the marked bundles and visit ALL their edges in the same order they were visited by the above traversal. The bundles visited are labeled B_1, B_2, ..., and the edges are labeled e_1, e_2, ...
  The number of edges to be forwarded from P_i is
    $\hat{F}_i$ = the total number of edges in marked bundles emanating out of P_i;
  Define the function $t(i)$ as $(i-1) \bmod (d - \lfloor \frac{d}{l} \rfloor) + 1$;
  /* The message associated with bundle B_i will be forwarded at time $t(i)$. */
  Let $r_i = d - (G_i - \hat{F}_i)$, the maximum number of edges that may be forwarded to P_i;
  Define $R_i = \sum_{j=1}^{i-1} r_j$;
  for i=1 **to** n **do**
    if $R_{i-1} \neq R_i$ **then** define $g(h) = i$ for each edge labeled e_h, and $R_{i-1} + 1 \leq h \leq R_i$;
  endfor
  /* Edge e_i will be forwarded to processor $g(i)$ */
  $(\hat{I}, \hat{G}) \leftarrow (I, G)$ minus all the edges in marked bundles of G;
  /* The edges in $\hat{G}$ will be added to reflect the forwarding operation. */
  for every processor P_j in G **do**
    for every marked bundle B_i emanating out of P_j in G **do**
      Let $S = \{g(l) | e_l \in B_i$ (note that each edge e_j is in a marked bundle)$\}$;
      Schedule X at time $t(i)$ will multicast the message associated with bundle B_i from processor P_j to the set of processors S (if $|S| = 1$, the operation is unicasting);
      for every edge $e_l \in B_i$ that is in a marked bundle **do**
        Add to the hold set of processor $g(l)$ (i.e., $H(g(l))$, in $\hat{G}$ message B_i;
        Add the edge from $P_{g(l)}$ to P_q in $\hat{G}$, where q is the processor where edge e_l ends in G;
      endfor
    endfor
  endfor
  Split every multi-edge bundle in $(\hat{I}, \hat{G})$ into single-edge bundles;
end of Procedure

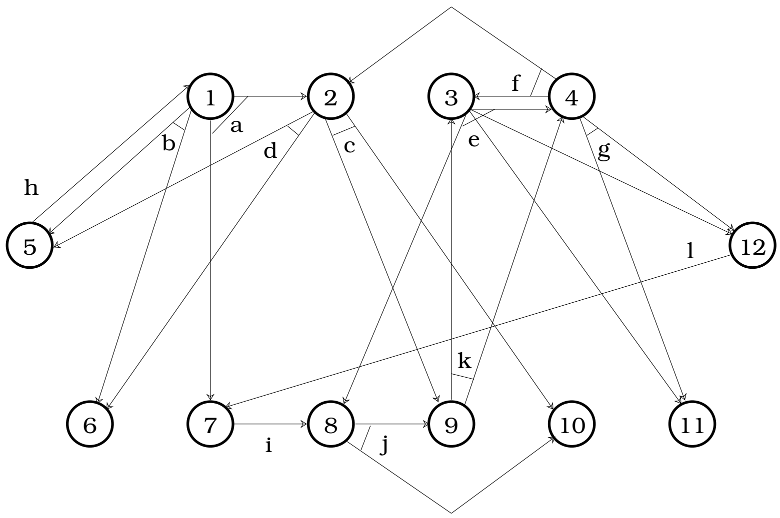

Figure 7: $l - MM_C$ problem instance $(I,\, G)$.

We now apply our algorithm to the problem instance given in Figure 7. The first loop transforms the problem to the problem instance given in Figure 8. The bundles that are split are the ones emanating out of processors P_3, P_8, and P_9.

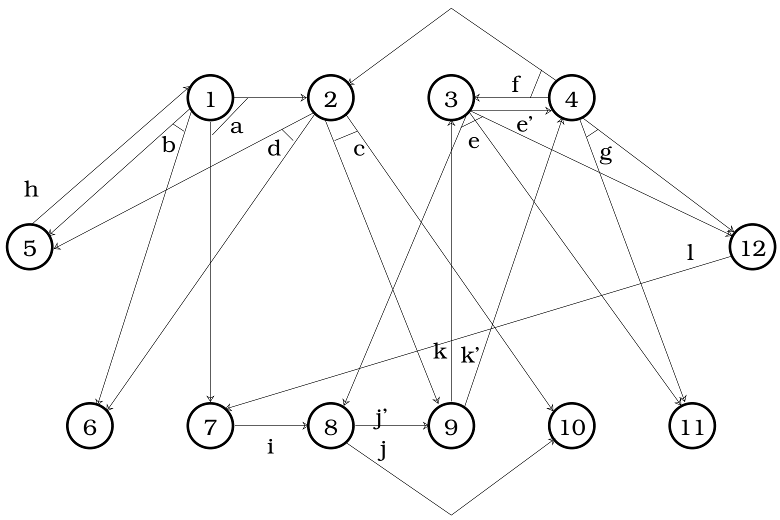

Figure 8: Problem instance $(I,\, G)$ after the first loop transformation.

In Figure 9 we show all the labelings performed by procedure l-FORWARD. The $\hat{F}_i$ and r_i values computed by the procedure are the numbers that appear on top of the vertices in Figure 9. The top number just below each of the vertices is the index of B_i, the next line shows the message name, and then the value $t(i)$. The line just above the bottom one has the index of the e_j, and the bottom number is the processor index to where that edge is to be forwarded.

The forwarding is: message a is multicasted to processors P_3 and P_5; message c is unicasted to processor P_6; message e is unicasted to processors P_7 and P_{10}; and message f is unicasted to processor P_{11}. The resulting problem, $(\hat{I}, \hat{G})$ is given in Figure 10. The difference now is

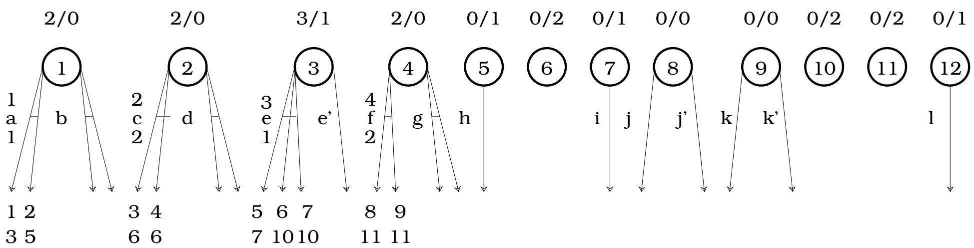

Figure 9: Labeling performed by procedure l-FORWARD.

that message a is to be transmitted form processor P_3 to P_2, and from P_5 to P_7; message c is to be multicasted from P_6 to P_9 and P_{10}; message e is to be unicasted from P_7 to P_8 and multicasted from P_{10} to P_{11} and P_{12}; and message f is to be multicasted from P_{11} to P_2 and P_3. In the final transformation all the multi-edge bundles are replaced by single-edge bundles. In our examples the bundles for messages b, c, e, f, and g are replaced by two edge single-edge bundles.

We should point out that all the $t()$ values could be set to one in the above example and there would not be any conflicts and the total communication time needed by l-FORWARD would be decreased from two to one. In general this is not always possible. For example if processor P_{10} had one edge emanation out of it then processor P_3 would only forward one edge there and the other one would be forwarded to processor P_{11}. Processor P_4 would also need to forward an edge to P_{11}. Therefore, the two messages to be received by P_{11} would need to be sent at different times, so the above reduction in total communication time is not possible in this other problem instance.

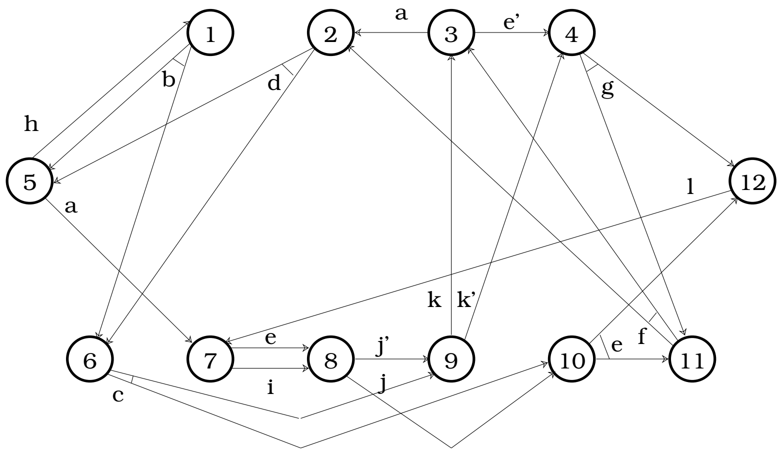

Figure 10: Resulting problem instance $(\hat{I}, \hat{G})$ just before last transformation (last line).

Theorem 3.1 *Problem instance $(\hat{I}, \hat{G})$ an instance of the MU_C problem and communication schedule X with total communication time $d - \lfloor \frac{d}{l} \rfloor + 1$ constructed by procedure FORWARD plus any communication schedule for $(\hat{I}, \hat{G})$ is a communication schedule for (I, G).*

Proof: First we show that schedule X is a feasible schedule. From the definition of F_i and $\hat{F}_i$ we know that only those processors with more than d edges will be forwarding edges to other processors, and the tentative number of such edges is $G_i - d$. Since the edges to be forwarded are the ones from the bundles with the largest number of edges, it then follows that all the edges to be forwarded belong to at most $d - \lfloor \frac{d}{l} \rfloor$ of the bundles emanating out of the processor. From the definition of $t()$ we know that these messages will be multicasted at different times. Let α be the total number of edges emanating out of processor P_i and let k be then number of single-edge bundles emanating out of P_i. The total number of edges in multi-edge bundles is $\alpha - k$, and this value is greater than or equal to $\alpha - d$ (the tentative number of edges to be forwarded) since $k \leq d$. This together with the way we define $\hat{F}_i$ implies that all the edges to be forwarded belong to multi-edge bundles and all the edges in these multi-edge bundles will be forwarded. From the $g()$ labels and procedure l-FORWARD we know that each processor will receive at most d edges to be forwarded. Since the messages forwarded consist of at least two edges, except possibly for the one forwarded to the previous and to the next processor, it follows that at most $\lfloor \frac{d}{2} \rfloor + 1$ messages will be received by each processor. Since $\lfloor \frac{d}{2} \rfloor + 1 \leq d - \lfloor \frac{d}{l} \rfloor + 1$, these messages are labeled sequentially, it then follows that all of these messages arrive at different times and there are no conflicts.

From the function $g()$ and procedure l-FORWARD we note that each message is forwarded to the appropriate processor so that if we carry out all the communications given by the resulting problem instance $(\hat{I}, \hat{G})$, we also solve problem instance (I, G). Furthermore, the resulting problem is of degree d. Therefore X plus any communication schedule for $(\hat{I}, \hat{G})$ is a communication schedule for (I, G).

□

Lemma 3.1 *The time complexity for procedure l-FORWARD is $O(n+e)$, where e is the total number of edges in (I, G).*

Proof: Since the proof is similar to Lemma 2.1 it is omitted.

□

Theorem 3.2 *Communication schedule X generated by procedure l-FORWARD 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 $2d - \lfloor \frac{d}{l} \rfloor + 1$. The overall time complexity for our procedure is $O(r(min\{r, n^2\} + n \ log \ n))$, where $r \leq dn$.*

Proof: The proof is omitted since it is similar to the one of Theorem 2.2.

□

In order for the communication schedule to be executable by pr-dynamic networks one needs to perform all the multicasting operations to adjacent numbered processors. In Figure 9 we see that message a will be multicasted only processors P_3 and P_5, and message e to processors P_7 and P_{10}. Since no other processor will transmit at the same time to P_4, P_8, and P_9, then message a can be multicasted to processors P_3, P_4, and P_5, and message e to processors P_7, P_8, P_9, and P_{10}. By applying this type of transformation we can establish the following result.

Theorem 3.3 *Communication schedule X plus X' is a communication schedule with total communication time $2d$ for any pr-dynamic network.*

Proof: As in the previous section, the proof of this theorem follows form the fact that all multicasting messages can be sent to a set of adjacent processors.

□

4 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 processors 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 [11].

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 $p \in T$ if message s is in processor p. Figure 11 (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 11 (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 [11]. The algorithm finds a maximum matching in a set of bipartite graphs. The time complexity is $O(es^{1.5}t^{0.5}\ log\ \mathrm{s})$, 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 4.1 *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.*

Proof: The proof follows from the above discussion.

□

5 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 algo-

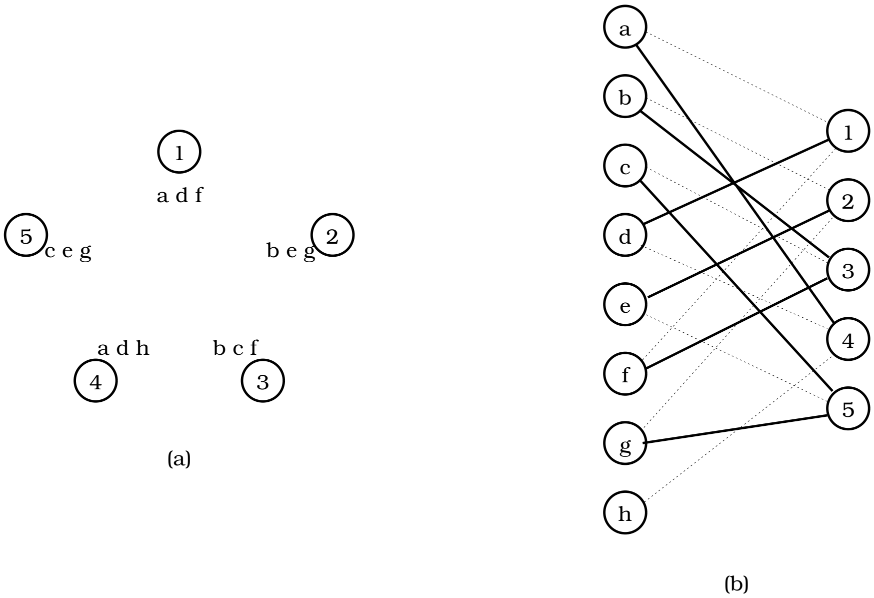

Figure 11: (a) MultiSource MMF_C problem instance. (b) Bipartite graph constructed from (a).

rithms ([6], [7]). However those algorithms are faster and were designed for the case when forwarding was not allowed. Our algorithm is a simplified version of the one in [9]. The approach we have taken 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).

The messages that need to be transmitted may be reduced as follows: processors in (I, G) with more than d emanating edges will keep d of their edges and forward the remaining ones, and processors with at most d emanating edges will keep all their edges. In our example only 10 edges are forwarded, rather than 23. We should point out that the new algorithm is more complex, but has the same approximation and time complexity bounds. This is the approach used by procedure l-FORWARD, and it results in a schedule with total communication time at most $\lfloor (2 - \frac{1}{l})d \rfloor + 1$, for the $l - MMF_C$ problem. For brevity we did not discussed the conditions under which one can delete the $+1$ from the above bound.

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), pp. 456 – 461, July 1996.

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

[9] T. F. Gonzalez, "Simple Multimessage Multicasting Approximation Algorithms Allowing Forwarding," Proceedings of the Tenth International Conference on Parallel and Distributed Computing Systems PDCS'97, (1997), 372 – 377.

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

[11] 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.

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

[13] 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.

[14] 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.

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

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

[17] 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.

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

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

[20] H. Shen, "Efficient Multiple Multicasting in Hypercubes," *Journal of Systems Architecture,* Vol. 43, No. 9, Aug. 1997.

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

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

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