Parallel Processing Letters, Vol. 19, No. 1 (2009) 129–139
© World Scientific Publishing Company

# Improved Communication Schedules with Buffers*

TEOFILO F. GONZALEZ
*Department of Computer Science, University of California, Santa Barbara, CA 93106-5110, USA*

Received June 2007
Revised July 2008
Communicated by K. Qiu

ABSTRACT

We consider the multimessage multicasting over the $n$ processor complete (or fully connected) static network when there are $l$ incoming (message) buffers on every processor. We present an efficient algorithm to route the messages for every degree $d$ problem instance in $d^2/l + l - 1$ total communication rounds, where $d$ is the maximum number of messages that each processor may send (or receive). Our algorithm takes linear time with respect to the input length, i.e. $O(n + q)$ where $q$ is the total number of messages that all processors must receive. For $l = d$ we present a lower bound for the total communication time. The lower bound matches the upper bound for the schedules generated by our algorithm. For convenience we assume that the network is completely connected. However, it is important to note that each communication round can be automatically translated into one communication round for processors interconnected via a replication network followed by a permutation network (e.g., two adjacent Benes networks), because in these networks all possible one-to-many communications can be performed in a single communication round.

*Keywords*: Approximation Algorithms, Multimessage Multicasting, Buffers, Networks, Communication Schedules.

## 1. Introduction[a]

Parallel and distributed systems were introduced to execute programs at unprecedented speeds. To accomplish this goal a program must be partitioned into tasks and the communications that must take place between these tasks must be identified to ensure a correct execution of the program. To achieve high performance one must assign each task to a processing unit (statically or dynamically) and develop communication programs to perform all the intertask communications efficiently. Efficiency depends on the algorithms used to route messages to their destinations,

*A preliminary version of this paper appeared in the Proceedings of the IASTED PDCS'05 Conference.
[a]Our introduction is a condensed version of our previous papers which include a complete justification for the multimessage multicasting problem as well as motivations, applications, and examples.

which is a function of the underlying communication network, the primitive operations and the communication model. Given a network with a communication model, a set of communication primitives and a set of messages that need to be exchanged, our problem is to find a schedule to transmit all the messages in the least total number of communication rounds. Generating an optimal communication schedule, i.e., one with the least total communication rounds, for our message routing problems over a wide range of communication networks is an NP-hard problem. To cope with intractability efficient message routing approximation algorithms for classes of networks under different communication assumptions have been developed. In this paper we consider the message communication problem where $l$ buffers have been placed at the receiving end of each processor. We show that speedups with factor of about $l$, over the case without buffers, can always be achieved.

The Multimessage Multicasting, $MM_C$, problem was introduced by Gonzalez [1, 2] and Shen [3]. The problem consists of constructing a communication schedule, for an $n$ processor static network (or simply a network), 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.

Each processor $P_i$ initially *holds* the set of messages $h_i$ and *needs* to receive the set of messages $n_i$. We assume that $\bigcup h_i = \bigcup n_i$, and that each message is initially in exactly one set $h_i$. We define the *degree* of a problem instance as $d = \mathbf{max}\{\mid h_i \mid, \mid n_i \mid\}$, i.e., the maximum number of messages that any processor sends or receives. Let $q$ be the total number of messages that all processors must receive. 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 [b]. The messages each processor holds and needs are given in Table 1. For this example the degree $d$ is 3 and $q$ is 18.

Table 1. Hold and Need Vectors for Example 1.1.

Hold Vector

| $h_1$ | $h_2$ | $h_3$ | $h_4$ | $h_5$ | $h_6$ | $h_7$ | $h_8$ | $h_9$ |
|---|---|---|---|---|---|---|---|---|
| $\{a, b\}$ | $\{c, d\}$ | $\{e, f\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |

Need Vector

| $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ | $n_8$ | $n_9$ |
|---|---|---|---|---|---|---|---|---|
| $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{a, c, e\}$ | $\{a, d, f\}$ | $\{b, c, e\}$ | $\{b, d, f\}$ | $\{c, d, e\}$ | $\{c, d, f\}$ |

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

One may 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 2 as a directed multigraph with additional thick lines that identify all edges or branches in each bundle.

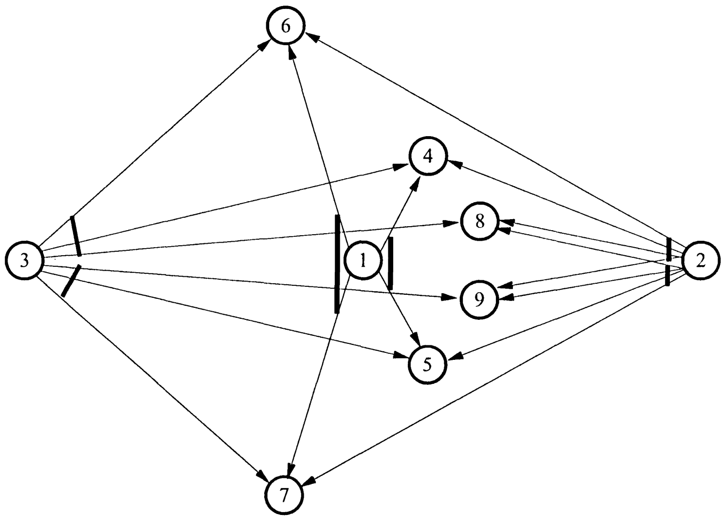

Fig. 1. Directed Multigraph Representation of Example 1.1. The thick line joins all the edges (branches) in the same bundle.

In the *single port communication mode* every processor sends at most one message and receives at most one message during each communication round. A processor may send at each communication round one of the messages it holds (i.e., a message in its hold set $h_i$ at the beginning of the time unit), but such message can be multicasted to any set of processors. The message also remains in the hold set $h_i$. During each time unit each processor may receive at most one message. The message that processor $P_i$ receives (if any) is added to its hold set $h_i$ at the end of the communication round.

The communication process ends when each processor has $n_i \subseteq h_i$, i.e., each processor holds all the messages it needs. The *total communication time* is the total number of communication rounds. 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.

Algorithms for the completely connected architecture have wide applicability in the sense that the schedules can be easily translated to communication schedules for every pr-network (MEIKO CS-2 machine), a large family of communication networks [2]. There is some penalty one has to pay for the translation process which is either doubling the communication rounds or extending the network. However, this penalty is not always incurred [4].

There are relatively inexpensive ways to speed-up communication. One such technique consists of adding $l$ buffers at the receiving end of each processor and developing controlling hardware so the buffering behaves as follows: (1) if at the beginning of a communication round one buffer has a message, then one such message (perhaps in a FIFO fashion) is passed to the processor and the buffer will be labeled empty for the current communication round; and (2) if there are $j$ empty buffers during the current communication round, then up to $j$ messages may be received and stored in these free buffers. 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 $d^2/l + l - 1$, where $d$ is the maximum number of messages that each processor may send (or receive) and $l$ is the number of input buffers on each processor. For $l = d$ we present a lower bound for the total communication time. The lower bound matches the upper bound for the schedules generated by our algorithm. For convenience we assume that the network is completely connected. However, it is important to note that each communication round can be automatically translated into one communication round for processors interconnected via a replication network followed by a permutation network (e.g., two adjacent Benes networks), because in these networks all possible one-to-many communications can be performed in a single communication round [2, 4].

## 2. Applications and Previous Results

The multimessage multicasting communication problem arises naturally when solving large scientific problems via iterative methods in a parallel or distributed computing environment, for example, solving large sparse system of linear equations using stationary iterative methods. Another application of multimessage multicasting arises when executing most dynamic programming procedures in a parallel or distributed computing environment. Dynamic programming procedures are heavily used in bioinformatics, operations research and computer science applications. In information systems, multimessage multicasting arises naturally when multicasting information over a $b$-channel ad-hoc wireless communication network. Other applications include sorting, matrix multiplication, discrete Fourier transform, etc. Message routing problems under the multicasting communication primitives arise in sensor networks. Other applications in high performance communication systems include voice and video conferencing, operations on massive distributed data, scientific applications and visualization, high performance supercomputing, medical imaging, etc. The need to deliver multidestination (multicasting) messages is expected to increase rapidly in the near future.

The case when each message has fixed *fan-out* $k$ (maximum number of processors that may receive any given message) has been studied [2]. For $k = 1$ (the problem is called multimessage unicasting $MU_C$), Gonzalez [2] showed that the problem corresponds to the makespan openshop preemptive scheduling problem (a generalization of the edge coloring of bipartite multigraphs) which can be solved in polynomial time, and each degree $d$ problem instance has a communication schedule with total communication time equal to $d$ [5].

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 [6, 7, 8, 9, 10, 11].

With the exception of the work reported in [1, 2, 3, 4, 12, 13, 14, 15], research has been limited to unicasting and most multicasting results are limited to single messages. Shen [3] has studied multimessage multicasting for hypercube connected processors. The heuristic try to minimize the maximum number of hops, amount of traffic, and degree of message multiplexing. Thaker and Rouskas [15] survey strategies for multimessage multicasting problems defined for all-optical networks.

Gonzalez [2] shows that a very restricted version of the $MM_C$ problem is NP-complete. Gonzalez [2] also shows that every degree $d$ instance of the $MM_C$ problem with $n$ processors has a communication schedule with total communication time at most $d^2$. This algorithm corresponds to our algorithm for the case when the number of buffers is 1. This bound is best possible in the sense that for all $d \geq 1$ there are problem instances that require $d^2$ communication time [2]. This is a lower bound holds the case when the number of buffers is 1.

When *forwarding* is allowed the problem is referred to as the $MMF_C$ problem. In this case messages may be sent through indirect paths even though single-edge paths exist. At first glance it appears that forwarding will not really help deliver messages faster for the $MM_C$ problem because forwarding consumes more resources, and the networks is complete (all the bidirectional links are present). The reduction used to establish that the $MM_C$ is NP-hard [2] can be easily modified to establish that the $MMF_C$ problem is an NP-hard problem [4]. Gonzalez [4] developed algorithms to construct communication schedules with total communication time at most $2d$ in $O(r(min\{r, n^2\} + n \log n))$ time, where $r$ is the total number of message destinations (therefore, $r \leq dn$). Clearly, these approximation algorithms are slower than the ones discussed above; however, they generate communication schedules with significantly smaller total communication time.

The *distributed* version of the $MMF_C$, which we refer to as the $DMMF_C$ problem has also been studied. An algorithm that performs all the communications in $O(d + \log n)$ expected communication steps is given in [13].

It is simple to see that the $DMMF_C$ problem is more general than the $MMF_C$ and the $MM_C$ problems, but the best communication schedule for the $DMMF_C$ problem has total communication time $\Omega(d + \log n)$ where as for the $MM_C$ problem is $d^2$, and just $2d$ for the $MMF_C$ problem. Therefore, knowing all the communication information ahead of time allows one to construct significantly better communication schedules, and forwarding plays a very important role in reducing the total communication time in our message routing problems. However, forwarding in all-optical communication systems is very expensive as it requires expensive optical to electrical to optical conversion of signals.

## 3. Algorithms and Lower Bound

We present a linear time algorithm to construct for every degree $d$ problem instance a communication schedule with total communication time at most $d^2/l+l-1$, where $d$ is the maximum number of messages that each processor may send (or receive) and $l$ is the number of input buffers on each processor. For $l = d$ we present a lower bound for the total communication time. The lower bound matches the upper bound for the schedules generated by our algorithm.

When $l = 1$ our algorithm generates a schedule with the same communication time as the one for the $MM_C$ problem. On the other hand when $l = d$ our algorithm generates a schedule with slightly smaller communication time than the one for the $MMF_C$ problem. Our new algorithm has the added property that it does not forward any of the messages and the time complexity to generate the solution is considerably smaller. Forwarding requires heavier link traffic since messages will be sent through more than one link. In other words, our new algorithm utilize the minimum amount of network capacity, but it requires buffers.

We assume that $d$ is a multiple of the number of buffers, $l$. First we define the set of $d^2/l$ colors as follows: $\{(i,j)|1 \leq i \leq d \text{ and } 1 \leq j \leq d/l\}$. Now assign an order $(1 \leq i \leq d)$ to all the bundles emanating from each vertex. In Figure 2 we show a problem instance with 9 processors and of degree $d = 6$. To simplify the figure the top three nodes do not receive messages and the bottom three processors do not send messages. Messages are represented by edges and multidestination messages are drawn with all the edges joined together by a horizontal line segment, for example the two edges emanating from node $A$ and ending at vertices $D$ and $E$.

Each outgoing edge from a processor is assigned a label which is just the index assigned to the bundle where it emanates, i.e., an integer value between 1 and $d$. The numbers near the origin of the edges in Figure 2 represent the label given to the bundles emanating from each vertex. The labeling of the bundles is an arbitrary one consistent with the labeling in our algorithm.

We order all the incoming edges to each processor $i$ in ascending order of their labels. This can be easily done via radix sort in $O(d+d_i)$ time, where $d_i$ is the number of incoming edges to processor $i$, since the labels are integers in the range $[1, d]$. Now with respect to this order assign the value of 1 to the first $l$ incoming edges to each processor, the value of 2 to next $l$ incoming edges to each processor, $\ldots$, and the value of $d/l$ to last $l$ incoming edges to each processor. For the example in Figure 2 the edges incoming to vertex $D$ have labels $3, 4, 1, 4, 1, 3$ as we traverse the node in clock-wise order starting at its left side. So the algorithm assigns the values $1, 2, 1, 2, 1, 2$ to these edges. Note that another possible choice would have been $2, 2, 1, 2, 1, 1$. Either one is consistent with the algorithm. All the values assigned to the edges in Figure 2 are drawn closer to the end of each directed edge.

Assign color $(i, j)$ to edge $e = \{p, r\}$ if $e$ belongs to the $i^{th}$ bundle emanating form vertex $p$, and $e$ is assigned the value of $j$ as an incoming edge to vertex $r$. Now we construct a schedule with total communication time $d^2/l + l - 1$ as follows. All the messages colored $(i, j)$ are transmitted at time $i + (j - 1)d$, for $1 \leq i \leq d$ and $1 \leq j \leq d/l$. In Figure 2 these transmission times are inside a small circle touching the corresponding edge. For example an edge from vertex $A$ to vertex $B$ is colored $(3, 1)$ and it is transmitted at time 3, where the two other edges between these two nodes are labeled $(4, 2)$ and $(1, 1)$ and are transmitted at time 10 and 1, respectively. Note the all the multidestination messages are transmitted at the same time except for two, one emanating out of vertex $B$ and the other out of vertex $E$.

Since for each processor there are at most $l$ messages incoming with the color $(i, 1)$ it then follows that none of the buffers for the processors will overflow from time 1 to time $d$. But there may be $l$ messages arriving at some processor at time $d$. In the following theorem we show that the buffers will not overflow when the remaining messages arrive. The algorithm overlaps in a clever way the process of emptying the buffers after all the messages colored $(i, j)$ and the messages $(i, j + 1)$ are sent for all $1 \leq j < d/l$. For example processor $E$ receives its messages at times $1, 11, 8, 1, 8, 1)$. Three messages are received at time 1, but the buffers will be empty before the next message arrives at time 8.

Our formal algorithm is listed below.

```
Procedure Ordered-Coloring
  Define the set of $d^2/l$ colors as $\{(i, j) | 1 \leq i \leq d$ and $1 \leq j \leq d/l\}$;
  for each processor $P_j$
    Assign an order $(1 \leq i \leq d)$ to all the bundles emanating out of processor $j$;
    Assign label $i$ to all the edges in the $ith$ bundle of processor $j$;
  endfor
  for each processor $P_j$
    Order all incoming edges to each processor $j$ in ascending order of their labels;
    With respect to this order assign the value of 1 to the first $l$ incoming edges to
      each processor, the value of 2 to next $l$ incoming edges to each processor, ...,
      and the value of $d/l$ to last $l$ incoming edges;
  endfor
  Each edge $e$ is assigned color $\{p, r\}$, where $p$ is its label and $r$ is its value as
    defined above;
  Our schedule $S$ transmits at time $i + (j - 1)d$, for $1 \leq i \leq d$ and $1 \leq j \leq d/l$,
    all the edges colored $(i, j)$;
end of Procedure Order-Coloring
```

**Theorem 1.** Procedure Ordered-Coloring described above generates a communication schedule with total communication time at most $d^2/l + l - 1$ for every degree $d$ instance of the $MM_C$ when there are $l$ buffers. Furthermore, the algorithm takes linear time with respect to the number of nodes and edges in the multigraph.

**Proof.** The proof follows the same arguments discussed above, except that we need to show that there is no overflow of buffers when we send the messages $(i, j)$ for $1 \leq i \leq d$ and $1 \leq j \leq d/l$. All the messages colored $(i, 1)$ are transmitted at time $i$, for $1 \leq i \leq d$. Since for each processor there are at most $l$ incoming messages colored $(i, 1)$, it then follows that none of the buffers for the processors will overflow. But it may be that for one or more processors there are $l$ messages that arrive at time $d$. Now, we need to show that that the messages colored $(i, 2)$ that are sent at time $i$ for $d + 1 \leq i \leq 2d$ will not overflow the buffers. Since the proof for the case $(i, j)$ for larger values of $j$ is very similar, we only prove the case for $j = 2$.

Consider the case when one is sending the messages colored $(1, 2)$ and let us consider any processor $Q$ that receives any subset of those messages. Clearly all of these messages belong to bundle 1 emanating out of some processors. All of the

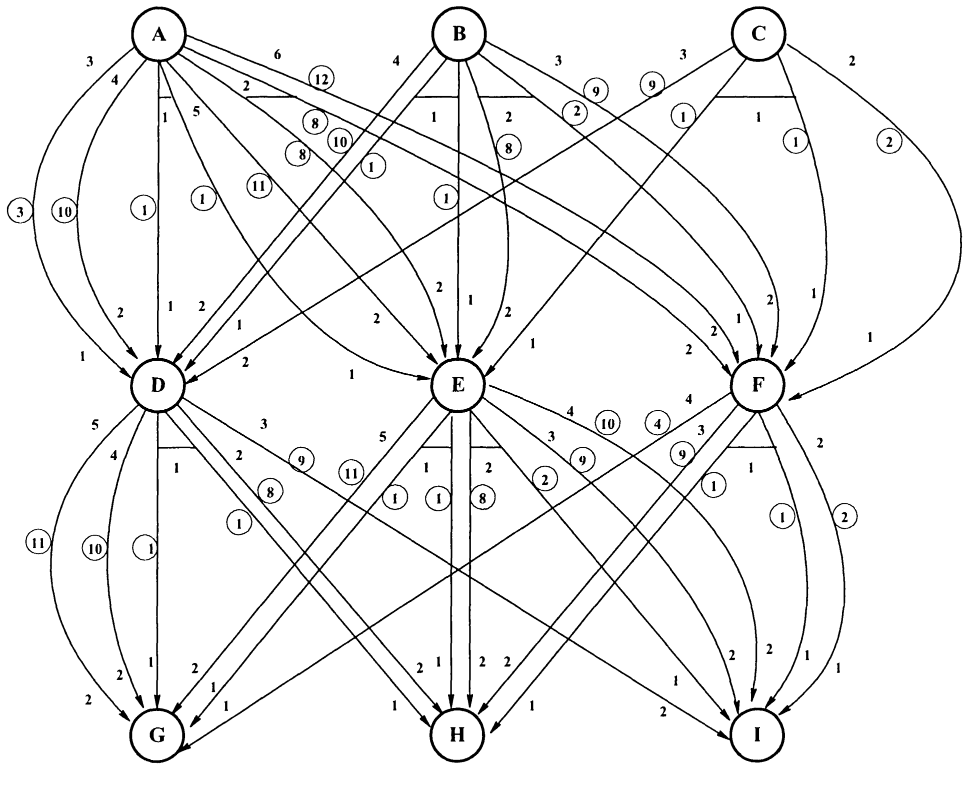

Fig. 2. Solution (with 3 buffers, i.e., $l = 3$) to problem instance with $d = 6$ and $n = 9$.

incoming messages to processor $Q$ were ordered according to their labels which are just the index of their bundles where they were sent from. Since the messages colored $(1, 2)$ have label 1, it then follows that all the messages received by processor $Q$ during the first $d$ communication rounds were colored $(1, 1)$. But that was $d$ time before messages $(1, 2)$ were sent, so the buffers are empty now since $l \leq d$. The same argument can be applied to all the messages colored $(i, 2)$. Since the proof of the cases when $j > 2$ is similar, we conclude the proof of the theorem.

The time complexity for the algorithm is bounded by the time required to color all the messages plus the total number of processors (since some processors send messages, but do not receive any message). The reason for this is that one can output a schedule listing the time at which every message is to be sent to its specific destination. Now, when describing the coloring algorithm we said we need to radix sort the incoming edges to each processor according to their labels. Since the labels have integer values in the range $[1, d]$ it follows that via radix sort this can be implemented to take $O(d + d_i)$ time for each processor $i$, where $d_i$ is the number of incoming edges to processor $i$. The overall time complexity bound would then be $O(nd + q)$. When $nd$ is about $q$ it is linear time algorithm. But when $nd$ is significantly larger than $q$ it is not a linear time algorithm. However, the coloring can be implemented to take $O(n + q)$ time by simply taking all the bundles labeled 1 and adding to the destination processor the appropriate label for the incoming edge. Since the addition is sorted by label, it follows that the whole list for each processor will be sorted. Since we need to check for messages emanating out of all processors and once we have processed all the bundles emanating out of a processor we do not need to check that processor again, we know that the time complexity bound is $O(n + q)$. □

The proof of the lower bound for the case when $l = 1$ that matched our upper bound is given in [2]. We now establish a lower bound for the total communication time of any schedule when $l = d$. Consider Example 3.1 given below.

**Example 3.1.** For any integer $d > 1$, we define problem instance $I_d$ as follows. The number of processors is $n = d^d + d$. Processors $P_1, P_2, \ldots, P_d$ send messages and do not receive any messages. The remaining processors $P_J$ for $J = (j_1, j_2, \ldots j_d)$, where each $j_l \in [1 : d]$, i.e., each $j_l$ is an integer whose value is between 1 and $d$, just receive messages. Each processor $P_J$ for $J = (j_1, j_2, \ldots j_d)$, receives the $j_l$th message bundle of processor $P_l$.

For the problem instance given in Example 3.1 we know that processor $P_1$ must send one of its messages for the first time at time $d$ or later simply because there are $d$ messages that processor $P_1$ must be send and no two of these messages may be sent concurrently. The same holds $P_2, P_3, \ldots, P_d$. Let $j_l$ be the message that processor $P_l$ sends last. Let $J = (j_1, j_2, \ldots, j_d)$. Now processor $P_J$ receives a message from all these processors at time $d$ or later. Since only one message may be received at a time, it requires $d - 1$ time units to receive all these messages. Therefore, the total communication time for every schedule must be at least $2d - 1$. It then follows that the total communication time is at least $2d - 1$.

## 4. Discussion

We have shown the buffers, a relatively inexpensive ways to speed-up communication, can be used to generate solutions that require considerable smaller total communication time than that required for the $MM_C$ problems. The solutions are similar to the ones obtained for the $MMF_C$ problems. However they can be generated much faster than for the $MMF_C$ problem. Furthermore, the solutions presented in this paper use the fewest number of communication links since messages are sent directly, rather than indirectly though several links. The most important open problem is to determine whether or not buffers can be used to reduce the total communication time for the $MMF_C$ problem to obtain communication schedules with at most $3d/2$ communication rounds.

We have established tight lower bounds when $l = 1$ and $d$. We conjecture that our solutions are also tight for all values of $l$.

## References

[1] T. F. Gonzalez, MultiMessage Multicasting, in *Proceedings The Irregular'96 Workshop* LNCS (1117), (Springer 1996) 217–228.

[2] T. F. Gonzalez, Complexity and Approximations for Multimessage Multicasting, *J. of Parallel and Distributed Computing* **55(2)** (1998) 215–235.

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

[4] T. F. Gonzalez, Simple Multimessage Multicasting Approximation Algorithms With Forwarding, *Algorithmica* **29** (2001) 511–533.

[5] T. F. Gonzalez, and S. Sahni, Open Shop Scheduling to Minimize Finish Time, *Journal of the ACM* **23(4)** (1976) 665–679.

[6] E. J. 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) 744–780.

[7] J. Whitehead, The Complexity of File Transfer Scheduling with Forwarding, *SIAM J. on Computing* **19(2)** (1990) 222–245.

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

[9] B. Hajek and G. Sasaki, Link Scheduling in Polynomial Time, *IEEE Transactions on Information Theory* **34(5)** (1988) 910–917.

[10] I. S. Gopal, G. Bongiovanni, M. A. Bonuccelli, D. T. Tang, and C. K. Wong, An Optimal Switching Algorithm for Multibeam Satellite Systems with Variable Band Width Beams, *IEEE Transactions on Communications* **30(11)** (1982) 2475–2481.

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

[12] T. F. Gonzalez, Improved Approximation Algorithms for Multimessage Multicasting, *Nordic Journal on Computing* **5** (1998) 196–213.

[13] T. F. Gonzalez, Distributed Multimessage Multicasting, *Journal of Interconnection Networks* **1(4)** (2000) 303–315.

[14] T. F. Gonzalez, On Solving Multimessage Multicasting Problems, *International Journal of Foundations of Computer Science* **12(6)** (2001) 791–808.

[15] D. Thaker and Rouskas, G., Multi-Destination Communication in Broadcast WDM Networks: A Survey, *Optical Networks* **3(1)** (2002) 34-44.