DISTRIBUTED MULTIMESSAGE MULTICASTING

Teofilo F. Gonzalez, University of California, Santa Barbara, CA

Abstract

We consider multimessage multicasting with forwarding over the n processor complete static network when each processor only knows the messages it needs to send and their destinations. We present an efficient distributed algorithm to route the messages with total expected communication time $O(d+\log n)$, where d is the maximum number of messages that each processor may send (or receive). Our routing algorithm consists of three phases: (1) processors exchange messages to learn basic global information; (2) each processor forwards its messages to transform the problem to a multimessage unicasting problem of degree d; and (3) a well known distributed algorithm is used to transmit all the unicasting messages.

Introduction

There are n processors, $P = \{P_1, P_2, \ldots, P_n\}$, interconnected via a fully connected network. Each processor is executing processes, and these processes are exchanging messages that must be routed through the network. We assume that processors alternate between computation and communication in a synchronous way. The Multimessage Multicasting problem, MM_C, consists of finding specific times when messages are to be transmitted so that all the communications can be carried in the least total time. When forwarding is allowed the problem is called the MMF_C problem. *Forwarding*, means that messages may be sent through indirect paths even though a single link direct paths exist, allows communication schedules with significantly smaller total communication time. In this paper we study the *distributed* version of the MMF_C, which we refer to as the $DMMF_C$ problem. In this the problem each processor initially knows the value of n and d, plus the messages it will be sending and their destinations. The non-distributed (or off-line) version is simpler because there is a preprocessing phase where all the information is available in one processor and it is used to construct communication schedules that are subsequently distributed to the individual processors. Hereafter we assume that each of the (original) messages to be transmitted is at least n bits long. This assumption allow us to send messages with length at most n, other than the original ones, and just count and report the total number of messages.

Let us formally define our problem. Each processor P_i *holds* the set of messages h_i and for each of its messages $m_{i,j}$ it knows the set of processors $s_{i,j}$ that must receive the message. From this information one can compute for each processor P_i the set of messages it *needs* to receive, n_i. Our algorithm does not compute the n_is, but at the end each processor P_i will have all the messages it needs. We define the *degree* of a instance as $d = \max\{|\ h_i\ |, |\ n_i\ |\}$, i.e., the maximum number of messages that any processor sends or receives. Consider the following example.

Example 1. Processors P_1, P_2, and P_3 send messages, and the remaining six processors receive messages [1]. The hold vector is: $h_1 = \{a, b\}, h_2 = \{c, d\}, h_3 = \{e, f\}, h_4 = h_5 = h_6 = h_7 = h_8 = h_9 = \emptyset$, and the need vector is: $n_1 = n_2 = n_3 = \emptyset$, $n_4 = \{a, c, e\}, n_5 = \{a, d, f\}, n_6 = \{b, c, e\}, n_7 = \{b, d, f\}, n_8 = \{c, d, e\}, n_9 = \{c, d, f\}$. The density d is 3, and $n = 9$.

Problem instances may be visualized via 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 is depicted in Figure 1 as a directed multigraph with additional thick lines that identify all edges or branches in each bundle.

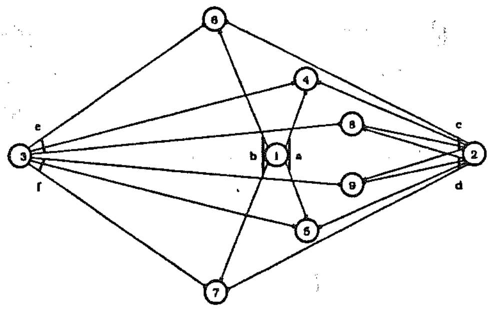

Figure 1: Directed Multigraph Representation for Example 1.

The communications allowed in our complete network for the distributed version of the problem must satisfy the restrictions given below. For the non-distributed case [1, 2, 3], rule 2 given below is simpler because those algorithms made sure that each processor received at most one message at a time. We should also point out that the last part of rule 2 is only needed for the third phase of our procedure, solving the resulting multimessage unicasting problem, because all the communications in the first two phases are predicatable, and communication conflicts can be avoided.

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. If two or more messages are sent to a processor at a time period, then the messages are garbled and the processor does not receive any of the messages. The sending processor will know at the end of time period whether or not the message it sent reached all its destinations. Note that if the message does not reach all its destinations, then the processor will not know the processors that received the message.

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

The process ends when $n_i \subseteq h_i$ for all i, i.e., each processor holds all the messages it needs. Our communication model allows us to transmit any of the messages in one or more stages. The *total communication time* is the latest time at which there is a communication.

The multimessage unicasting problem MU_C is exactly like the multimessage multicasting problem, except that each message must be sent to exactly one destination. An efficient algorithm for the multimessage unicasting problem MU_C is given in [1]. Several papers (see [1]) have reported contributions to related problems. The distributed version of the multimessage unicasting problem with forwarding, $DMUF_C$, has been studied in the context of optical-communication parallel computers [4, 5, 6, 7]. Valiant [7] presented a distributed algorithm with $O(d+\log n)$ total expected communication time. The algorithm is based in part on the algorithm by Anderson and Miller [4]. The communication time is optimal, within a constant factor, when $d = \Omega(\log n)$, and Gereb-Graus and Tsantilas [5] raised the question as to whether a faster algorithm for $d = o(\log n)$ exits. This question was answered in part by Goldberg, Jerrum, Leighton and Rao [6] who show all communication can take place in $O(d + \log\log n)$ communication steps with high probability, i.e., if $d < \log n$ then the failure probability can be made as small as n^α for any constant α. Gereb-Graus and Tsantilas [5] presented distributed algorithms without forwarding with $\Theta(d + \log n \log\log n)$ expected communication steps. With the exception of the work reported in [1, 2, 3], research has been limited to unicasting and single message multicasting.

Gonzalez [1] showed that even when $k = 2$ the decision version of the MM_C and the MMF_C problem are NP-complete, presented an efficient algorithm to construct for any degree d problem instance a schedule with total communication time at most d^2, and presented problem instances for which this upper bound is best possible. The lower bound holds when the number of processors and the fan-out is huge. Since this situation is not likely to arise in the near future, results for the MM_C problem with restricted fan-out has been reported in [1]. Gonzalez [2, 3] present efficient algorithms to construct for every degree d instance a schedule with total communication time at most $2d$, where d is the maximum number of messages that each processor may send (receive). These algorithms 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 is a degree d multimessage unicasting problem. The second phase generates a communication schedule for this problem by reducing it to the Makespan Openshop Preemptive Scheduling problem which can be solved in polynomial time.

In this paper we present an algorithm for the $DMMF_C$ problem, which combines the classic parallel prefix algorithm, a distributed version of the message forwarding phase of Gonzalez' algorithm [3] for multimessage multicasting with complete information, and Valiant's [7] distributed algorithm for the multimessage unicasting problem. The result is a distributed algorithm that routes all messages in $O(d + \log n)$ expected communication steps.

Approximation Algorithm

Our strategy is to use the classic parallel prefix algorithm to compute and exchange information, and then use this information to run a distributed version of Gonzalez' algorithm [3] with partial global information. By forwarding all the messages, Gonzalez' algorithm [3] transforms the problem to a multimessage unicasting problem. All of the resulting communications can be performed by Valiant's [7] distributed algorithm.

Before we proceed it is important to understand the message forwarding phase of Gonzalez' algorithm [3] that reduces the problem to a multimessage unicasting problem. Let us explain how this phase works by applying it to the problem instance given in Figure 2. The problem instance consists of 12 processors, 11 messages, and has degree $d = 2$.

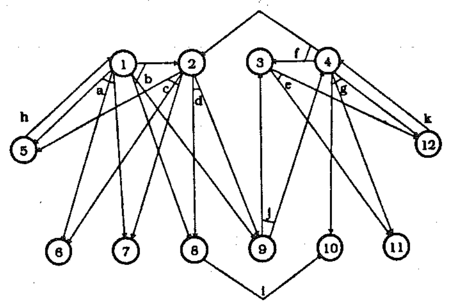

Figure 2: MM_C problem instance (I, G).

In Figure 3 we show all the processors with a list of labels assigned to the bundles and edges that are defined as follows. The top set of numbers is the bundle number which is defined by labeling the bundles emanating out of processor P_1, then the one emanating out of P_2, and so forth. The next label is the message for the bundle and the third one is the bundle number modulo (d) plus 1. This third number is the time at which the message associated with the bundle will be forwarded. From the way these labels are generated, we know that no two bundles emanating out of a processor will forward a message at the same time. The edges are labeled beginning with the ones emanating out of the first bundle, then the second one, and so forth. These labels are shown in the fourth line. The last set of numbers is the ceil of the edge number divided by d. This last row indicates the processor index where the message will be forwarded. It is simple to see that each processor will receive at most d messages and all these messages will be received at different times.

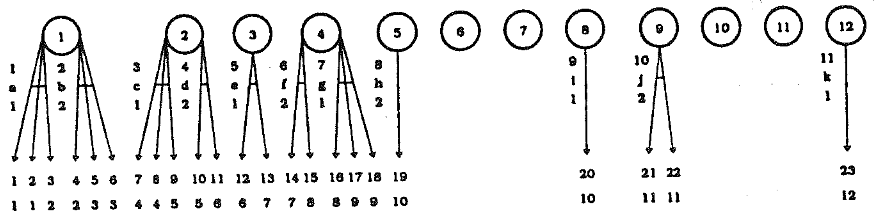

Figure 3: Labeling performed by procedure FORWARD.

The specific communication operations for time 1 and 2 are given in the forests labeled $T1$ and $T2$ in Figure 4. The resulting unicasting problem $(\hat{I}, \hat{G})$ of degree d is given in Figure 4 (all objects in (c)). Remember that Gonzalez' algorithm [3] is non-distributed and all the information is known globally. However, the algorithm in this paper is distributed and the only information every processor initially knows are the messages it needs to send and their destinations, the values of n and d. Let us now explain the three steps of our algorithm.

1. **Compute and Broadcast Basic Information.** Each processor P_i needs to know the total number of messages that processors $P_1, P_2, \ldots, P_{i-1}$ need to send as well as the total number of destinations for all of these messages. I.e., the total number of bundles and the total number of edges emanating out of all of these processors. This information is needed

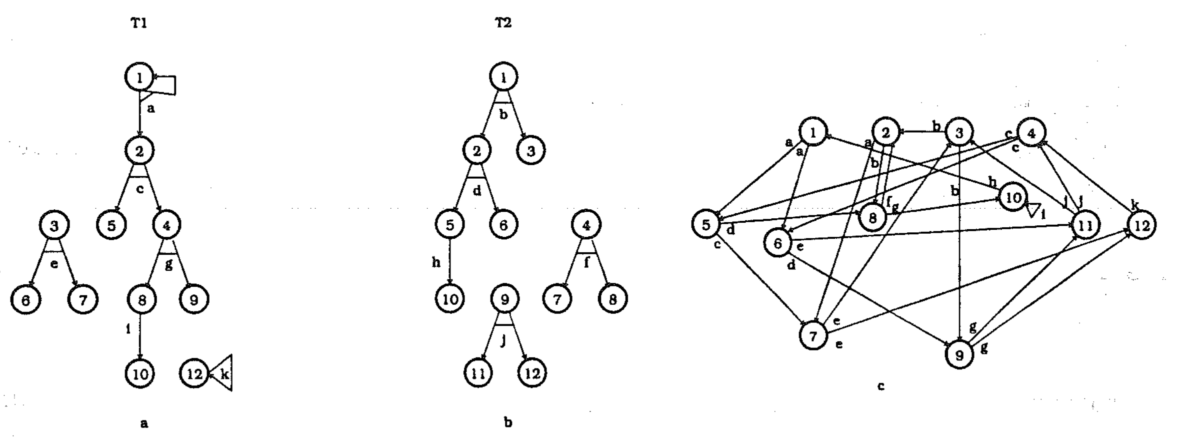

Figure 4: Communications at time one (a) and time two (b). In (c) the MU_C problem instance $(\hat{I},\hat{G})$ constructed from (I,G) in Figure 2.

to label the bundles and edges emanating out of P_i, and it can be easily computed via the classic parallel prefix in $O(\log n)$ communication steps.

2. **Transform to the Multimessage Unicasting Problem.**

   This operation is performed by making Gonzalez' algorithm [3] distributed. Each processor will run the following procedure.

```
Procedure FORWARD for $P_j$
    /* The values of $n$ and $d$ are known in every processor */
    /* The following information computed in Step 1 is in processor $P_j$
       $n_b$: total number of bundles emanating out of $P_1, P_2, \ldots, P_{j-1}$.
       $n_e$: total number of edges emanating out of $P_1, P_2, \ldots, P_{j-1}$. */
    Label $B_{n_b+i}$ the $i^{th}$ bundle visited while traversing the bundles emanating from $P_j$;
    Define $t(n_b+i)$ as $(n_b+i-1) \bmod (d)+1$;
    /* The message associated with bundle $B_{n_b+i}$ will be forwarded at time $t(n_b+i)$. */
    Label $e_{n_e+i}$ the $i^{th}$ edge visited while traversing the bundles emanating out of $P_j$
       in the order $B_{n_b+i}, B_{n_b+i+1}, \ldots$;
    Define the function $g(n_e+i)$ as $\lceil \frac{n_e+i}{d} \rceil$;
    /* Edge $e_{n_e+i}$ will be forwarded to processor $P_{g(n_e+i)}$ */
    for every bundle $B_{n_b+i}$ emanating out of $P_j$ do $S_{n_b+i} \leftarrow \{g(l) | e_l \in B_{n_b+i}\}$; endfor
    for $t = 1, 2, \ldots, d$ do
      if there is a bundle emanating out of $P_j$ with $t(n_b+i) == t$ then
          At time $t$ processor $P_j$ multicasts message $B_{n_b+i}$ to the set of processors $S_{n_b+i}$ (if
             $|S_{n_b+i}| = 1$, the operation is unicasting). Send a bit vector of size $n$ appended
             to this message indicating the indices of the processors that must eventually
             receive this message, the first processor that will receive this forwarded
             message and the number of edges that such processor must forward.
             /* This info is used by the forwarding processors to compute the destinations
             of the messages being forwarded. */
    endfor
end of Procedure
```

3. **Solving the resulting Multimessage Unicasting Problem Instance.**

   At this point each processor just runs Valiant's algorithm [7] and all the messages are delivered to their destinations in $O(d + \log n)$ expected communication steps.

Lemma 1. The pair $(\hat{I}, \hat{G})$ is a problem instance of the MU_C problem and once all its messages are transmitted will solve the original multimessage multicasting problem (I, G).

Proof. The proof of the lemma is based on the observations that the $t()$ and $g()$ labels computed by our procedure are identical to the ones that Gonzalez' algorithm [3] computes. This implies that the messages will be forwarded exactly as in Gonzalez' algorithm [3]. Therefore, solving the resulting multimessage unicasting problem solves the original problem. □

Theorem 1. Our algorithm performs all the multicasting for every instance of the $DMMF_C$ problem with $O(d + \log n)$ expected communication steps.

Proof. The proof is based in the previous lemma, and the correctness proofs for the subprocedures used by our algorithm. The total number of communication steps in phase 1 is $O(\log n)$, and in phase 2 is $O(d)$. The expected number communication steps for phase 3 is $O(d + \log n)$. □

The most important open problem is to develop distributed algorithms with similar performance guarantees for processors connected via a dynamic networks. Algorithms exist for the non-distributed version of this problem [3]. The main difficulty in extending that work to the distributed case is the construction of the routing tables with only local information.

References

[1] T. F. Gonzalez, "Complexity and Approximations for Multimessage Multicasting," *Journal of Parallel and Distributed Computing,* 55, 215 – 235, 1998.

[2] T. F. Gonzalez, "Algorithms for Multimessage Multicasting With Forwarding," Proceedings of the 10^{th} PDCS, 372 – 377, 1997.

[3] T. F. Gonzalez, "Simple Multimessage Multicasting Approximation Algorithms With Forwarding," UCSB TRCS-97-24, December 1997.

[4] R. J. Anderson, and G. L. Miller, Optical Communications for Pointer Based Algorithms, TRCS CRI 88 – 14, USC, Los Angeles, 1988.

[5] M. Gereb-Graus and T. Tsantilas, "Efficient Optical Communication in Parallel Computers," Proceedings of 4th SPAA, 1992, pp. 41 – 48.

[6] L. A. Goldberg, M. Jerrum, T. Leighton, and S. Rao., "Doubly Logarithmic Communication Algorithms for Optical-Communication Parallel Computers," *SIAM J. Comp.*, 26, No. 4, (1997), pp. 1100 – 1119.

[7] L. G Valiant, "General Purpose Parallel Architectures," Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, New York, 1990, Chapter 18 (see p 967).

ANALYSIS AND DESIGN OF PROTOCOLS STACKS AT INTELLIGENT NETWORK'S NODE

Adam Grzech, Wrocław University of Technology, Wrocław, Poland

Introduction

The paper is devoted to present and discuss a proposed model of so-called intelligent computer communication network's node as composed of protocols set [1,2]. In such active networks, one of the primary functions of the communication subsystem is to provide the transfer of an incoming traffic from a source node to a destination node using set of protocols adequate to required quality of services in changing environments. The proposed model of network's source node may be used also to model transfer of traffic in any nodes. It is applied to formulate and solve protocols stack analysis, design and optimisation tasks [2,4]. Processing times and increasing rates associated with every pair of protocols are utilised to formulate performance criteria describing quality of ordered protocols set [2].

A protocols stack in intelligent network's node is equivalent to existence of particular routing path between a source and a destination protocols. The quality of services depends on protocols traversed by the incoming traffic as well as on environment in which the input traffic is served [2,5].

Network Node Model

A network's node is modelled as graph with subset of protocols as graph nodes and pair of co-operating protocols as graph links. The protocols set is composed of separate subsets where subsets of source protocols and destination protocols are distinguished. The subsets contain protocols with similar scope of functionality and different value of parameters describing quantity of delivered services. An existence of link between ordered pair of protocols from separate subsets means that the second protocol is required to complete services assured by the first in assumed protocols stack. In order to evaluate the quality of protocols stack, two positive values are associated with every two ordered and linked protocols. The first is a delay generally measures the desirability of using a particular pair of linked protocols, with a lower delay meaning more desirable. The second value denotes an average increase of traffic; the protocol operating at the upper level serves some amount of incoming traffic and produces larger amount of traffic that is served by next protocol operating at the lower level.

A network node is modelled as graph with protocols set P and links set L. The protocols set P is composed of $N+1$ subsets P_n where $n=0,1,...,N$. P_0 contains $|P_0|$ source protocols while P_N contains $|P_N|$ destination protocols [2,5]. The stack of protocols is modelled as a route between the source and destination protocols. An existence of link between ordered pair of protocols p_{nl} and p_{mj} ($p_{nl} \in P_n$, $p_{mj} \in P_m$, $n<m$ and $n,m=0,1,...,N$) means the first protocol (p_{nl}) is supported by the second protocol (p_{mj}) or in other words, the *j-th* protocol from the *m-th* subset delivers services for the *l-th* protocol from *n-th* subset. It is assumed that the protocols are applied to compose stack of protocols co-operating within layered and nested architecture. In order to evaluate the quality of protocols stack, a cost $c(p_{nl}, p_{mj})$ is associated with every two ordered and linked protocols p_{nl}