# An Efficient Algorithm for Gossiping in the Multicasting Communication Environment

Teofilo F. Gonzalez, *Member*, *IEEE*

**Abstract**—We present an algorithm for the gossiping problem defined over an $n$ processor communication network, $N$, where message multicasting is allowed. The algorithm generates a communication schedule with a total communication time at most $n + r$, where $r$ is the radius of the network. Our algorithm begins by constructing a spanning tree (or tree network $T$) with the least possible radius. Then, all the communications are carried out in the tree network as follows: Each processor waits its turn to transmit "almost" consecutively to its parent and children all the messages in its subtree. During other times, each processor transmits to its children all the messages emanating elsewhere in the network.

**Index Terms**—Approximation algorithms, gossiping, multicasting, communication schedules.

## 1 Introduction

Let $N$ be any communication network (or graph) with $n \geq 3$ processors (nodes or vertices). The *broadcasting* problem defined over $N$ consists of sending a message from one processor in the network to all the remaining processors. The *gossiping* problem (also known as the all-to-all broadcasting problem) over $N$ consists of broadcasting $n$ messages, each originating at a different processor. Gossiping problems have been studied under many different objective functions and communication models. Our communication model allows each processor to multicast one message to any subset of its adjacent processors, but no processor may receive more than one message at a time. 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. Multicasting is a powerful communication primitive that allows for the communications to be performed much faster than when restricting to the *telephone* (or *unicasting*) *communication* model (a processor may transmit a message to just one of its adjacent processors at a time) or the *broadcasting communication* model (a processor may transmit a message to all the adjacent processors). Consider the problem instance given in Fig. 1, where an optimal solution to the gossiping problem can be easily constructed as follows: In the first communication round, each processor sends to its clockwise neighbor the message it holds, and then, in the remaining iterations, every processor transmits to its clockwise neighbor the message it just received from its counter-clockwise neighbor. The total communication time is $n - 1$, which is best possible.

Let us formally define our problem. Initially, each processor $P_i$ *holds* one message in its hold set $h_i$ and *needs* to receive the remaining $n - 1$ messages. The multicasting communications model in our network must satisfy the following two restrictions:

1. During each time unit $t$, each processor may receive at most one message provided such message was sent during the previous time unit $(t - 1)$. The message that processor $P_i$ receives (if any) at time $t$ is added to its hold set $h_i$ at time $t$.
2. During each time unit $t$, each processor $P_i$ may transmit one of the messages it holds (i.e., a message in its hold set $h_i$ at time $t$), and such a message can be transmitted simultaneously to a set of processors adjacent to $P_i$. The message will also remain in the hold set $h_i$. It is important to note that the receive operation is performed before the send operation. In other words, a message that is sent to $P_i$ at time $t - 1$ arrives at $P_i$ at time $t$ and processor $P_i$ may send it at time $t$ to other processors.

The communication process ends when each processor has the $n$ messages. The above communication rules define a communication round (or step) for a communication schedule as follows: A *communication round* $C$ 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 round $C$ must obey the following communications rules imposed by our network at each step:

1. Every pair of $D$ sets in $C$ are disjoint, i.e., every processor receives at most one message; and
2. All the indices $l$ in $C$ are distinct, i.e., each processor sends at most one message.

If at time $t$ processor $P_i$ receives a message, then the message was sent to $P_i$ during communication round $t - 1$. The message is added at time $t$ to $h_i$ and it may be sent from $P_i$ during communication round $t$ or later.

A *communication schedule* is a sequence of communication rounds. A communication schedule $S$ that solves the gossiping problem for a given graph $G$ is a communication schedule such that, after performing all the communication rounds, every processor will hold the $n$ messages. The *total communication time* is the number of communication rounds

• *The author is with the Department of Computer Science, University of California, Santa Barbara, CA 93106. E-mail: teo@cs.ucsb.edu.*

*Manuscript received 4 Oct. 2001; revised 27 Aug. 2002; accepted 3 Jan. 2003. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 115142.*

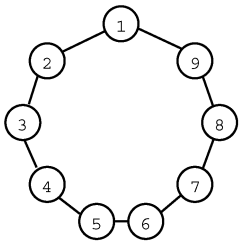


Fig. 1. Network ($N_1$) with a Hamiltonian circuit.

in schedule $S$, which is identical to the latest time there is a communication. Our problem consists of constructing a communication schedule with the least total communication time. From the communication rules, we know that, in every problem instance, every processor needs to receive $n-1$ messages and, since no processor may receive two or more messages simultaneously, it follows that $n-1$ is a trivial lower bound on the total communication time. Therefore, the schedule we constructed for network $N_1$ is an optimal one. In order to make communication rounds equal in time, we say that the first round of communications is sent at time zero and received at time one, the second round is sent at time one and received at time two, and so on.

In this paper, we are mainly concerned with the **offline** gossiping problem, i.e., the schedule is constructed by a processor that knows all the information about the problem ahead of time. We discuss the **online** problem in the final section, where the $n$ processors know limited global information.

The solution we constructed for network $N_1$ suggests a method for solving the gossiping problem. The idea is to first construct a Hamiltonian circuit and then use that circuit to transmit all the messages in $n-1$ time units. It is well-known that determining whether or not a graph has a Hamiltonian circuit is an NP-complete problem and it is conjectured that there is no efficient algorithm for its solution [10]. Fortunately, it is not necessary for a network to have a Hamiltonian circuit in order for the gossiping problem be solvable in $n-1$ steps. There are networks that do not have a Hamiltonian circuit, but in which gossiping can be performed in $n-1$ communication steps even under the telephone communication model. The Petersen graph (Fig. 2) is one such graph.

Network $N_3$ (Fig. 3) does not have a Hamiltonian circuit, but gossiping can be performed in $n-1$ communication steps under the multicasting communication model, but not

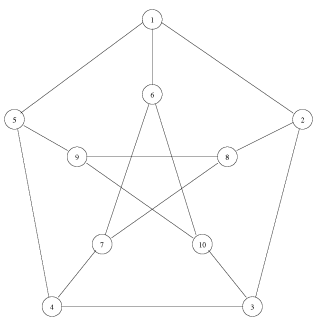


Fig. 2. Network $N_2$ (Petersen graph).

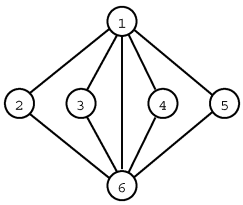


Fig. 3. Network ($N_3$).

under the telephone communication model [16]. Since the telephone communication model is a restricted version of the multicasting communication model, the above example establishes that multicasting is a much more efficient way to communicate.

The above examples suggest that it is always possible to perform gossiping in our communication model in $n-1$ steps. However, this is not always the case. Consider a straight line network with three processors. After the first round of communications takes place, the center processor will know at most two messages because it can only receive one message at a time. Therefore, it is not possible to deliver to one of the two ends all the messages in two time units. Now, consider a straight line network with $n = 2m+1$ processors, for any positive integer $m$. The earliest time that all the $n$ messages can arrive at the center processor is $n-1$. The earliest time at which the last message that arrived at the center processor can be received by both of the end processors is $n-1+m$. Therefore, every communication schedule for this problem instance has a communication time at least $n+r-1$, where $r$ is the radius of the network. In this paper, we present an algorithm for the gossiping problem that generates a schedule with $n+r$ communication steps for any network.

## 2 Previous Work, New Results, and Applications

The broadcasting and gossiping problems are not new. These problems have been studied for the past three decades [7], [17]. However, most of the work is for the telephone type of communication, i.e., at every step, each processor may transmit information to at most one processor and no processor may receive information from more than one processor at a time. Some of the previous work allows for up to $n$ messages to be transmitted over a single link in one time unit. In other words, the transmission packets may be of size $\Omega(n)$, which implies that such algorithms are not scalable. In this paper, as well as in the more traditional ones, we limit the amount of information to be transmitted over a link during one time unit to one message. Under the traditional model these problems are computationally difficult, i.e., NP-hard. But, there are efficient algorithms to construct optimal communication schedules for restricted networks under some communication models [5], [9], [19], [21]. Up to now, there is no known polynomial time approximation algorithm with constant approximation ratio for the broadcasting problem defined over arbitrary graphs, i.e., there is no known efficient approximation algorithm $A$ such that $f(\hat{I})/f^*(I) \leq c$ for every problem instance $I$,

where $f(\hat{I})$ is the total communication time for the schedule constructed by algorithm $A$ for problem instance $I$, $f^*(I)$ is the total communication time of an optimal schedule for problem instance $I$, and $c$ is a constant. Determining whether or not such an algorithm exists has been an intriguing open problem for more than two decades. The best known approximation algorithms appear in [1], [18], [21], and a randomized algorithm is presented in [6].

Broadcasting under our communication model is trivial to solve. At time zero, the processor that has the message broadcasts it to all its neighbors. Then, at each iteration, each processor that just received a message will plan to multicast it to all its neighbors that do not have the message. But, if there are two or more processors currently planning to send a processor the message, then only one of them will actually send it. This is trivial to solve because we are dealing with the offline scheduling problem. Once the round of communications is completed, we start with the next iteration. It is simple to see that every processor $i$ receives a message at time $j$ if and only if the shortest path (remember that all edges have weight one) from the broadcasting processor to processor $i$ has $j$ edges. Clearly, the total communication time in the communication schedule generated by the above procedure is equal to the maximum length of a shortest path from the broadcasting processor to a processor in the network. The above algorithm is clearly offline.

A variation of the gossiping problem in which there are costs associated with the edges and there is a bound on the maximum number of packets that can be transmitted through a link at each time unit has been studied in [8]. Approximation algorithms for several versions of this problem are given in [2], [8], [11].

Routing under the multicasting communication model has been considered in [12], [13], [14], [20]. In this problem, each processor needs to transmit a set of messages, but each message is to be received by its own subset of processors. Shen [20] studied the problem for hypercube connected processors, and Gonzalez [12], [13], [14] considered the problem for fully connected processors and also for processors interconnected via a multistage interconnection network that satisfies some simple properties (e.g., the MEIKO CS-2 parallel computer system). The gossiping problem is a restricted version of the multimessage multicasting problem; however, all the previous algorithms for the multimessage multicasting problem are for a set of architectures. The algorithm for the gossiping problem in this paper works for any arbitrary network.

In this paper, we study the gossiping problem under the multicasting communication model. Our main motivation for using this communication model is that it has been available for many years and it allows us to generate solutions with fewer communication steps than when using the telephone communication model. For example, the Meiko CS-2 allows multicasting, and one may build arbitrary networks using components similar to the ones in the Meiko machine. Synchronization may be achieved through different techniques, one such technique is through software barriers. The ability of processors to send information concurrently to more than one destination (which we call multicasting) arises naturally in wireless communications where a transmission with power $r^\alpha$ reaches all receivers at a distance $r$, where $\alpha$ is between two and four [4]. In static sensor networks, the traditional message routing problems take an interesting new twist where, in addition to having to disseminate information quickly, one also desires to maximize the battery life of the units [4]. Our communication model also arises naturally in optical networks. Gossiping arises in many application [3], [19] that include sorting, matrix multiplication, Discrete Fourier Transform, solving linear equations, etc.

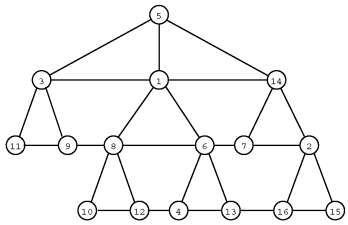


Fig. 4. Network. (Numbers indicate the processor number.)

In Section 3, we begin by reducing the network gossiping problem to gossiping in a tree with least possible height (or depth). Then, we give a simple but inefficient algorithm to construct a communication schedule for any arbitrary tree. Our main result are two algorithms, `Propagate-Up` and `Propagate-Down`, that, when executed concurrently, generate a communication schedule for the gossiping problem with communication time $n + r$, where $n$ is the number of processors and $r$ is the network radius (or height of the tree).

## 3 Algorithms

In this section, we discuss our algorithm to generate a communication schedule with total communication time at most $n + r$, where $r$ is the network radius. The *radius* of a network is the least integer $r$ such that there is a vertex $v$ at a distance at most $r$ from every vertex in the graph. For example, there is a path from every vertex in the graph to vertex $v$ with at most $r$ edges. Our procedure consists of two steps. First, we build a special tree network (Section 3.1) and, then, we perform all the communications in that tree network (Section 3.2).

### 3.1 Constructing the Tree Network

As we mentioned before, the first step of the algorithm constructs a minimum-depth spanning tree. Such a tree can be easily constructed by performing $n$ breadth-first search (BFS) traversals of the graph starting at each vertex and then selecting the tree with least height (or depth). This procedure takes $O(mn)$ time. All the communications will be performed in the resulting tree network.

Applying the above procedure to the network in Fig. 4, results in the network given in Fig. 5. In the next section, we present our algorithms for gossiping in trees.

### 3.2 Gossiping in Tree Networks

The problem of gossiping in an arbitrary network has been reduced to gossiping in a tree with height $r$. Our algorithm for gossiping in trees is a little bit complex, so, before we discuss it, we introduce two procedures that generate communication schedules with a larger total communication time. Let us begin by defining some terms. The topmost

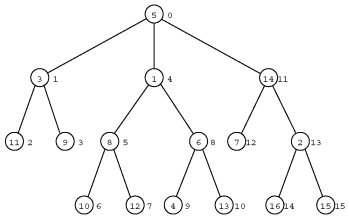

Fig. 5. Tree Network generated from the graph in Fig. 4. (Numbers outside the circles indicate the message label.)

vertex is called the *root* of the tree. The *level* of every vertex in the tree network is defined as follows: The level of root is zero, the level of the children of the root is equal to one, the grandchildren are at level 2, and so forth. For every vertex, fix the ordering of the subtrees in any arbitrary order. Our algorithm proceeds by labeling the message originating at each vertex in depth-first search order starting with the one at the root (label 0) and ending at some leaf (label $n-1$). Applying the algorithm to the problem tree in Fig. 5, we obtain the labels that appear to the right of the vertices.

We begin by discussing the first algorithm (`Simple`) to perform the gossiping in $2n+r-3$ time units. This procedure has been used to solve other message routing problems. The idea is to send up all the messages to the root of the tree first in such a way that message $i \geq 1$ is received by the root at time $i$. The message labeled $i$ at level $k$ is transmitted to its parent (if any) at time $i-k$, to its grandparent (if any) from its parent at time $i-k+1$, and so on. Clearly, there are no conflicts and at time $n-1$, all the messages are received by the root. Clearly, this process takes $n-1$ communication steps. Now, all the messages need to be propagated downwards. At time $n-2$, message 0 is sent from the root to all its children; at time $n-1$, message 1 is sent from the root to all its children; and so on. Note that; during this entire process, when a nonroot vertex receives a message from its parent, it immediately multicasts it to all its children. It is simple to verify that, by time $2n+r-3$, all processors will receive all the messages. The main advantage of procedure `Simple` is that it is quite simple, but on the other hand, the total communication time of the schedules it generates is not so small. This is a problem when we have to perform a large number of gossiping operations using the same tree network.

**Lemma 1.** *The communication schedule generated by procedure* `Simple` *has a total communication time* $2n+r-3$ *for any tree with* $n$ *processors and height* $r$.

**Proof.** The proof follows from the above discussion. □

Procedure `UpDown`, developed by Gonzalez [15], is more complex, but the communication schedule it generates has a smaller total communication time. The approach is similar to procedure `Simple`, except that, at the same time, the algorithm sends messages up and down throughout the tree. The procedure consists of two phases. In the first phase, like in the algorithm `Simple`, all the messages are propagated to the root, but, at the same time, it begins the process of propagating messages to other parts of the tree. In the second phase, the algorithm just propagates down some messages that got stuck in the network. The first and second phase take $n-1+r$ and $2(r-1)+1$ steps, respectively.

In this paper, we introduce algorithm `ConcurrentUp-Down`, which is rather simple and it generates the best schedules with respect to the total communication time. It is based on the observation that all the operations can be carried out in a single stage by maximizing the concurrent operations performed at each step. The total communication time of the schedules generated by this algorithm is just $n+r$. Our algorithm is a simplified version of our previous algorithm [16], and its correctness proof is much simpler.

Let us now introduce additional notation. Consider vertex $v$ at level $k \geq 1$. Vertex $v$ has message $i$ initially and the subtree rooted at $v$ includes message $i$ up to message $j$ initially. The parent of nonroot vertex $v$, which we refer to as $v'$, has message $i'$ initially and the subtree rooted at vertex $v'$ includes message $i'$ up to message $j'$ initially. The level of vertex $v'$ is $k'=k-1$. Remember that vertices are labeled from 0 to $n-1$.

The messages in every nonroot vertex $v$ are labeled as follows:

- Messages $0,1,\ldots,i-1$ and $j+1,j+2,\ldots,n-1$ are called the *other* messages or *o-messages*.
- Messages $i,i+1,\ldots,j$ are called the *body* messages or *b-messages*. The b-messages are partitioned with respect to vertex $v$ as follows:
  - Message $i$ is called the *starting* message or *s-message* at $v$.
  - Message $i+1$, if $i+1 \leq j$, is called the *lookahead* message or *l-message* .
  - Messages $i+2,\ldots,j$ (if any) are called the *remaining* messages or *r-messages*.
- The b-messages are also partitioned with respect to vertex $v'$, (the parent of vertex $v$) as follows:
  - Message $i$, if $i=i'+1$, is the *lookahead in parent (lip)* message or *lip-message*.
  - Messages $max\{i,i'+2\},\ldots,j$, if any, are the *remaining in parent (rip)* messages or *rip-messages*.

Note that the root of the tree ends up labeled as follows: Message $i=0$ is the s-message, message 1 is the l-message, and messages $2,\ldots,n-1$ are called r-messages. All messages are rip-messages and there are no lip-messages.

First, we present algorithm Propagate-Up to construct a schedule that indicates when all the messages are transmitted to the root of the tree. Then, we present algorithm Propagate-Down that indicates when all the messages are propagated down to all the vertices. Since there will be no conflicts between these two schedules, then the overlap of the two schedules is our final schedule. We refer to the procedure that invokes Propagate-Up and Propagate-Down and then combines the schedules generated by these procedures as algorithm *ConcurrentUpDown*. After performing all the communications in the final schedule, every vertex will have the $n$ messages by time $n+r$.

In order to avoid conflicts with the schedule generated by `Algorithm Propagate-Down`, the lip-message at each vertex $v$ will be sent by `Algorithm Propagate-Up` at time 0. If we do not do this and instead we were to modify `Algorithm Propagate-Down`, then some messages would get stuck at each level, like in the algorithm in [12], and the total communication time would be more than $n+r$. More specifically, consider node 1 (with message 4) in

Fig. 5. Suppose message 5 was not sent to processor 1 at time zero, but instead it was sent at the latest time which is time 3 (and, thus, received at time 4). Now, message 3 arrives at the root at time 3 and it is sent at that time to processor 1. Then, there would be a conflict (two different messages sent at the same time to processor 1). So, one needs to delay sending message 3 from the root. In other words, message 3 would get stuck in the root as in [12] and, hence, the total communication time would increase in many cases.

Every vertex $v$ labels its messages as defined above. Then, algorithms `Algorithm Propagate-Up` and `Algorithm Propagate-Down` are executed at each vertex $v$.

`Algorithm Propagate-Up (v)`

**(U1)** {Time 1}. If $v$ is not a leaf vertex (i.e., $i+1 \leq j$), then vertex $v$ receives message $i+1$ at time 1.

**(U2)** {Time $i-k+2, \ldots, j-k$}. Starting at time $i-k+2$, vertex $v$ receives sequentially from its children all its r-messages (if any) in increasing order of their message number. The first message is $i+2$, which is received at time $i-k+2$, and the last one, message $j$, is received at time $j-k$.

**(U3)** {Time 0}. If $v$ is not the root of the tree, then, at time 0, vertex $v$ sends to its parent its lip-message, if it has one.

**(U4)** {Time $i-k+w, \ldots, j-k$}. Let $w$ be the number of lip-messages at vertex $v$. If $v$ is not the root of the tree, then, starting at time $i-k+w$, vertex $v$ sends to its parent sequentially in increasing order of their message number all its rip-messages. The first of these messages is message $i+w$ and it is sent at time $i-k+w$, and the last message is message $j$ and it is sent at time $j-k$.

`End of Algorithm Propagate-Up`

**Lemma 2.** *Algorithm* `Propagate-Up` *is feasible, i.e., every vertex $v$ at level $k$ in the tree receives the messages in steps as specified in (U1) and (U2), and every nonroot vertex $v$ in the tree has the messages available when they need to be sent as specified in Steps (U3) and (U4). It is assumed that no other procedure sends messages that interfere with the ones sent by algorithm* `Propagate-Up`.

**Proof.** The proof is by induction on the height of the vertex $v$.

*Basis:* Vertex $v$ has height 1 (it is a leaf node). Clearly, the total number of messages originating in the subtree rooted at $v$ is one and it is the s-message $i$, which is available at vertex $v$ at time 0. Since there are no l-messages nor r-messages, it follows that the all these messages are received as specified in Steps (U1) and (U2).

There is nothing else to prove when vertex $v$ is the root of the tree. So, let's consider the case when $v$ is a nonroot vertex. Since message $i$ is the s-message at $v$, it is available at time zero and message $i$ is either a lip-message or a rip-message at $v$. So, if message $i$ is a lip-message, then it is possible to send it at time 0 as specified by Step (U3). On the other hand, if it is a rip-message, then it is scheduled to be delivered at time $i-k$. From the DFS labeling, we know that $i \geq k$, so $i-k \geq 0$. So, it follows that it is available to be sent as specified in Step (U4). This completes the proof of the base case.

*Induction Hypothesis:* Assume the lemma holds when $v$ has height $l-1 \geq 1$.

*Induction Step:* Show that the lemma holds when $v$ has height $l$.

By definition, the l-message at $v$ is a lip-message in exactly one subtree of $v$. Since every subtree of $v$ has a height less than $l$, then, by the induction hypothesis, we know that the lip-message $i+1$ in a child of $v$ is sent at time 0 from a child of $v$ to $v$. Since no other processor sends a message to $v$ at time 0, it follows that the l-message is received by $v$ as specified in Step (U1).

By definition, every r-message in $v$ is a rip-message in exactly one subtree of $v$. Since every subtree of $v$ has height less than $l$, then, by the induction hypothesis, we know that r-message $i+\alpha$, for $2 \leq \alpha \leq j-i$, in $v$ is sent at time $i+\alpha-(k+1)$ from a child of $v$ (remember that the child of $v$ is at level $k+1$) and, thus, received at time $i+\alpha-k$ by $v$. Therefore, each r-message is received starting at time $i+\alpha-k$. Since the first r-message, if any, is $i+2$, then $\alpha=2$ and the messages start arriving at time $i+\alpha-k=i+2-k$. Since the remaining r-messages are received sequentially, then Step (U2) can be correctly executed.

There is nothing else to prove when $v$ is the root of the tree. So, let us now consider the case when $v$ is a nonroot vertex. Since message $i$ is an s-message, it is available at time 0 at $v$ and it is either a lip-message or a rip-message.

If message $i$ is a lip-message, then it is available to be sent at time 0 and it can be sent as specified in Step (U3).

If message $i+\alpha$ for some $0 \leq \alpha \leq j-i$ is a rip-message at $v$, then, if it is an s-message, it will be available at time zero at $v$ if it is an l-message, then, since we have established Step (U1), it will be available at time 1 at $v$, and if it is an r-message, then, since we have established Step (U2), it will be available at time $i+\alpha-k$ at $v$. Since $i \geq 1$ and $i \geq k$ then, in all cases, it arrives at $v$ by time $i+\alpha-k$, which is the same time at which it will be sent to its parent. Since the remaining rip-messages will arrive sequentially in ascending order of their message number, we know Step (U4) can be correctly executed. □

`Algorithm Propagate-Down (v)`

If vertex $v$ is not the root of the tree, then perform Steps (D1) and (D2), and, if it is a nonleaf vertex, then also perform Steps (D2) and (D3).

**(D1)** {Time $2, \ldots, i-k+1$ and $j-k+3, \ldots, n+k$}. From time 2 to time $i-k+1$ and from time $j-k+3$ to time $n+k$ nonroot vertex $v$ receives all its o-messages from its parent.

**(D2)** {Time $2, \ldots, i-k-1$ and $j-k+1, \ldots, n+k$}. All the o-messages received as in Step (D1) by nonroot and nonleaf vertex $v$ are sent to all the children of $v$ by vertex $v$ at the same time they are received, except for the messages received at times $i-k$ and $i-k+1$, if any, which are sent at time $j-k+1$ and $j-k+2$. When $i=k$, then no messages are received by $v$ from its parent at times $i-k$ and $i-k+1$.

TABLE 1
Schedule for the Vertex with the Message Labeled 0 in Fig. 5

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Receive from Child | - | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | - |
| Send to Children | - | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 |

**(D3)** {Time $i-k,\ldots,j-k$}. From time $i-k$ to $j-k$, messages $i$ to $j$ are sent by nonleaf vertex $v$, in increasing order of their message number, to all its children, except to the child of $v$, which already has the message. The only message that it sends to all the children of $v$ is message $i$. When $i=k$, message $i$ is not sent at time $i-k$, but it is sent at time $j-k+1$.

`End of Algorithm Propagate-Down`

**Lemma 3.** *If Algorithm Propagate-Down is feasible, then Algorithm Propagate-Up is feasible, i.e., the root of the tree receives all its messages as in Lemma 2 (see Steps U1 and U2), then the messages are available to be sent as in Step (D3) and, in the remaining vertices, if the messages are received as specified by Step (D1) and Lemma 2 (see Steps (U1) and (U2)), then all the messages will be available to be sent as specified in Steps (D2) and (D3). It is assumed that no other procedure sends messages that interfere with the ones sent by algorithm* `Propagate-Down`.

**Proof.** Let vertex $v$ be the root of the tree (it is at level zero). By Lemma 2, we know that, at time $i=1,2,\ldots,n-1$, message $i$ will arrive at $v$ and that is precisely the time at which these messages will be sent to each child that does not already have the message, and message 0, which is available at time zero, will be sent at time $n$. Therefore, the root of the tree propagates the messages as in Step (D3).

The proof is by induction on the level ($\geq 1$) of vertex $v$.

*Basis:* Vertex $v$ is at level one (Vertex $v$ is a child of the root). Since the parent of vertex $v$ is the root, we know that Step (D3) in Propagate-Down sends from the root each b-message $i$ at time $i$ and it arrives at $v$ at time $i+1$, for $1\leq i\leq n-1$. Message 0 is sent at time $n$ and arrives at $v$ at time $n+1$. Therefore, all the o-messages will arrive at $v$ from its parent during the time intervals $2,\ldots,i-k+1$ and $j-k+3,\ldots,n+k$ as specified in Step (D1). Therefore, all these messages can be sent as specified in Step (D2). From Lemma 2, we know that messages $i,\ldots,j$ are available to be sent to all the children of $v$ as specified in Step (D3).

*Induction Hypothesis:* Assume the lemma holds when $v$ is at level $l-1\geq 1$.

*Induction Step:* Show that the lemma holds when $v$ is at level $l$.

Vertex $v$ is at level $l$ which is greater than two. Since the parent of vertex $v$ is not the root, we know that Step (D3) in Propagate-Down sends from the parent of $v$ all the o-messages and they arrive at $v$ as in Step (D1) because the parent of $v$ is at level $k-1$. Therefore, all the o-messages will arrive at $v$ from its parent during the time intervals $2,\ldots,i-k+1$ and $j-k+3,\ldots,n+k$ and all these messages can be sent as specified in (D2). From Lemma 2, we know that messages $i,\ldots,j$ are available to be sent to all the children of $v$ as specified in Step (D3).□

The schedules given in Tables 1, 2, 3, and 4 are for the vertices with the messages labeled 0, 1, 4, and 8 in Fig. 5. The schedule for the vertex with message labeled 0 is straight forward. Message $i$ is received at time $i$ and it is sent at time $i$. The schedule for the vertex with the message labeled 1 is simple since it receives the b-messages and then the o-messages.

The schedule for the vertex with the message labeled 4 is more complex since it includes messages 2 and 3 that are delayed. The schedule of the vertex with the message labeled 8, it is more complex since messages 6 and 7 are the ones delayed at the node.

TABLE 2
Schedule for the Vertex with the Message Labeled 1 in Fig. 5

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Receive from Parent | - | - | - | - | - | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 |
| Receive from Child | - | 2 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| Send to Parent | 1 | 2 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| Send to Child | - | 2 | 3 | 1 | - | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 |

TABLE 3
Schedule for the Vertex with the Message Labeled 4 in Fig. 5

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Receive from Parent | - | - | 1 | 2 | 3 | - | - | - | - | - | - | - | 11 | 12 | 13 | 14 | 15 | 0 |
| Receive from Child | - | 5 | - | - | - | 6 | 7 | 8 | 9 | 10 | - | - | - | - | - | - | - | - |
| Send to Parent | - | - | - | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | - | - | - | - | - | - | - |
| Send to Child | - | - | 1 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 2 | 3 | 11 | 12 | 13 | 14 | 15 | 0 |

Procedure `ConcurrentUpDown` just overlaps the schedules generated by algorithms `Propagate-Up` and `Propagate-Down`.

**Theorem 1.** *The communication schedule generated by procedure* `ConcurrentUpDown` *has a total communication time $n + r$ for any network with $n$ processors and radius $r$.*

**Proof.** The proof of this theorem follows from Lemmas 2 and 3 provided that we show that the assumption about the messages being sent by algorithms `Propagate-Up` and `Propagate-Down` not interfering with each other actually hold, and every vertex receives all the messages. Vertex $v$ receives messages at time 1 and at times $i - k + 2$ to $j - k$ and sends messages (if it is not the root) at time 0 and at times $i - k + w$ to $j - k$. On the other hand, procedure `Propagate-Down` receives messages at times 2 to $i - k + 1$ and $j - k + 3$ to $n + k$ and sends them out at times 2 to $i - k - 1$, $j - k + 1$ to $n + k$, and $i - k$ to $j - k$. Clearly, all the messages received by the two procedures are received at different times. The same holds for the ones sent by the procedures, except for the ones sent at times $i - k + w$ to $j - k$. But, from Steps (U4) and (D3), it is clear that the same messages are being sent at these times; therefore, since processors may send the same message to more than one processor, there is no conflict.

From Steps (U1) and (U2), it is easy to see that the root receives all the b-messages. Every leaf node receives all the o-messages because of Step (D1). The nonroot and nonleaf nodes receive all their b-messages ((U1), (U2)) and o-messages (D1). Therefore, the resulting schedule solves the gossiping problem. □

## 4 Discussion

We have presented an algorithm to construct communication schedules with total communication time at most $n + r$, where $r$ is the radius of the graph. The algorithm is efficient and generates near optimal solutions. By near optimal, we mean that it generates a schedule with total communication time at most 1.5 times that of an optimal one. This follows from the fact that the radius, $r$, of a network is at most $n/2$. The most time consuming part of the algorithm is finding a minimum-depth spanning tree which takes $O(mn)$ time. All the other Steps of the algorithm to construct the schedule take $O(n)$ time. In many applications, one has to execute the gossiping algorithms a large number of times, so that is why it is important to perform gossiping in a tree efficiently. The construction of the tree is performed only when there is a change in the network, which we assume remains constant for long periods of time.

A minimum-depth spanning tree for a straight line graph with an odd number of processors is a tree whose root is the center processor of the line graph and each of the two subtrees is just a straight line. When the graph has $2m + 1$ vertices, for any positive integer $m$, the radius of the graph and the tree is $m$. In Section 1, we showed that every schedule for such instance of the gossiping problem has total communication time greater or equal to $n + r - 1$. The one that our algorithm constructs is $n + r$. One may improve the performance of our algorithm by one unit, but the protocol for each processor will not be uniform and the algorithm will be much more complex. The reason is that one needs to alternate the delivery of messages from different subtrees.

TABLE 4
Schedule for the Vertex with the Message Labeled 8 in Fig. 5

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Receive from Parent | - | - | - | 1 | 4 | 5 | 6 | 7 | - | - | - | 2 | 3 | 11 | 12 | 13 | 14 | 15 | 0 |
| Receive from Child | - | 9 | - | - | - | - | - | - | 10 | - | - | - | - | - | - | - | - | - | - |
| Send to Parent | - | - | - | - | - | - | 8 | 9 | 10 | - | - | - | - | - | - | - | - | - | - |
| Send to Child | - | - | - | 1 | 4 | 5 | 8 | 9 | 10 | 6 | 7 | 2 | 3 | 11 | 12 | 13 | 14 | 15 | 0 |

Our algorithms can be easily adapted for the online case. The only global information that they need is the value of $i$, $j$, and $k$. Once this information is disseminated throughout the network, each processor may send its messages at the specified times. Our algorithm can also be easily adapted to the weighted gossiping problem where each processor has at least one message to transmit. The idea is to replace a processor that needs to send $l$ messages with a chain with $l$ processors. In practice, one only mimics this splitting process.

## REFERENCES


[1] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, "Multicasting in Heterogeneous Networks," *SIAM J. Computing,* vol. 30, no. 2, 2000.

[2] J.C. Bermond, L. Gargano, A.A. Rescigno, and U. Vaccaro, "Fast Gossiping by Short Messages," *SIAM J. Computing,* vol. 27, no. 4, pp. 917-941, 1998.

[3] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, 1989.

[4] O. Egecioglu and T. Gonzalez, "Minimum-Energy Broadcast in Simple Graphs with Limited Node Power," *Proc. IASTED Parallel and Distributed Computing Systems Conference,* pp. 334-338, 2001.

[5] S. Even and B. Monien, "On the Number of Rounds Necessary to Disseminate Information," *Proc. First ACM Symp. Parallel Algorithms and Architectures,* pp. 318-327, 1989.

[6] U. Feige, D. Peleg, P. Raghavan, and E. Upfal, "Randomized Broadcast in Networks," *Proc. Int'l Symp. Algorithms SIGAL* pp. 128-137, 1990.

[7] P. Fraigniaud and E. Lazard, "Methods and Problems of Communication in Usual Networks," *Discrete Applied Math.,* vol. 53, pp. 79-133, 1994.

[8] P. Fraigniaud and S. Vial, "Approximation Algorithms for Broadcasting and Gossiping," *J. Parallel and Distributed Computing,* vol. 43, pp. 47-55, 1997.

[9] S. Fujita and M. Yamashita, "Optimal Group Gossiping in Hypercube under Circuit Switching Rounds," *SIAM J. Computing,* vol. 25, no. 5, pp. 1045-1060, 1996.

[10] M.J. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman and Company, 1979.

[11] L. Gargano, A.A. Rescigno, and U. Vaccaro, "Communication Complexity of Gossiping by Packets," *J. Parallel and Distributed Computing,* vol. 45, pp. 73-81, 1997.

[12] T.F. Gonzalez, "Complexity and Approximations for MultiMessage Multicasting," *J. Parallel and Distributed Computing,* no. 55, pp. 215-235, 1998.

[13] T.F. Gonzalez, "Simple Algorithms for MultiMessage Multicasting with Forwarding," *Algorithmica,* vol. 29, pp. 511-533, 2001.

[14] T.F. Gonzalez, "Improved Approximation Algorithms for Multi-Message Multicasting," *Nordic J. Computing,* vol. 5, pp. 196-213, 1998.

[15] T.F. Gonzalez, "Gossiping with Multicasting Communication Primitives," *Proc. 2000 Parallel and Distributed Computing Systems Conference,* pp. 768-773, Nov. 2000.

[16] T. Gonzalez, "Gossiping in the Multicasting Communication Environment," *Proc. Int'l Parallel and Distributed Processing Symp.,* 2001.

[17] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman, "A Survey of Gossiping and Broadcasting in Communication Networks," *NETWORKS,* no. 18, pp. 129-134, 1988.

[18] J. Hromkovic, R. Klasing, B. Monien, and R. Peine, *Dissemination of Information in Interconnection Networks (Broadcasting and Gossiping).* D.Z. Du and D.F. Hsu, eds. Kluwer Academic, pp. 273-282, 1995.

[19] D.W. Krumme, K.N. Venkataraman, and G. Cybenko, "Gossiping in Minimal Time," *SIAM J. Computing,* vol. 21, no. 2, pp. 111-139, 1992.

[20] H. Shen, "Efficient Multiple Multicasting in Hypercubes," *J. Systems Architecture,* vol. 43, no. 9, Aug. 1997.

[21] R. Ravi, "Rapid Rumor Ramification," *Proc. 35th Ann. Symp. Foundations of Computer Science,* pp. 202-213, 1994.


**Teofilo F. Gonzalez** received the BSc degree in computer science from the Instituto Tecnologico de Monterrey (1972) and the PhD degree in computer science from the University of Minnesota, Minneapolis, (1975). He is currently a professor of computer science at he University of California, Santa Barbara. His research activity has concentrated on the development of efficient exact and approximation algorithms for problems in several areas, including job scheduling, parallel and distributed algorithms, CAD/VLSI placement and routing, and graph applications. Dr. Gonzalez' current research interests include: multimessage multicasting algorithms, routing algorithms, and approximation algorithms. He is a member of the IEEE and the IEEE Computer Society.