

AD-A156 049

June 1985

LIDS-P-1473

COMMUNICATION COMPLEXITY OF DISTRIBUTED SHORTEST PATH ALGORITHMS *

Baruch Awerbuch **

Robert G. Gallager **

ABSTRACT

A new distributed shortest path algorithm for an asynchronous communication network with unit edge weights is presented. The sum, over all network edges, of the number of elementary messages used in finding shortest paths from a given root to all other nodes is $O(V^{1.6} + E)$ where V is the number of nodes and E the number of edges. For dense networks, with $E \geq V^{1.6}$, this order of complexity is optimum.

This document has been approved for public release and sale; its distribution is unlimited.

DTIC
ELECTE
JUN 28 1985
S A D

* This research was conducted at the M.I.T. Laboratory for Information and Decision Systems with partial support provided by the National Science Foundation under Grant NSF-ECS-8310698 and by the Defense Advanced Research Projects Agency under Contract ONR/N00014-84-K-0357. Also support was provided to B. Awerbuch by a Weizmann Fellowship.

**Authors are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

85 6 17 107

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A156 049	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) COMMUNICATION COMPLEXITY OF DISTRIBUTED SHORTEST PATH ALGORITHMS	5. TYPE OF REPORT & PERIOD COVERED	
	6. PERFORMING ORG. REPORT NUMBER LIDS-P-1473	
7. AUTHOR(s) Baruch Awerbuch and Robert G. Gallager	8. CONTRACT OR GRANT NUMBER(s) DARPA Order No. 3045/2-2-84 Amendment #11 ONR/N00014-84-K-0357	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Massachusetts Institut of Technology Laboratory for Information and Decision Systems Cambridge, Massachusetts 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Code No. 5T10 ONR Identifying No. 049-383	
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209	12. REPORT DATE June 1985	
	13. NUMBER OF PAGES 32	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A new distributed shortest path algorithm for an asynchronous communication network with unit edge weights is presented. The sum over all network edges, of the number of elementary messages used in finding shortest paths from a given root to all other nodes is $O(V^{1.6} + E)$ where V is the number of nodes and E the number of edges. For dense networks, with $E \geq V^{1.6}$, this order of complexity is optimum.		

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special



INTRODUCTION

We consider an asynchronous communication network, modelled by a simple undirected graph, $G = (V, E)$. The nodes V model the communication processors of the network and the edges E model the bidirectional communication channels of the network. Each node receives and transmits messages on its adjacent edges and performs local computations. The sequence of messages sent in a given direction on an edge is received error free by the opposite node in first in first out (FIFO) order with finite but unpredictable delays. These messages are queued at the receiving node until the processor is ready to process them.

In a communication network, the amount of communication required to perform some function is often more significant than the amount of processing required, so the complexity of a distributed algorithm is appropriately measured in terms of the total number of bits communicated over all edges. Here, for simplicity, we simply count the number of elementary messages sent over all edges, where an elementary message contains at most some small fixed set of parameters, such as the size of a set of nodes or the length of a path. Thus the number of bits in an elementary message can grow at most logarithmically with the size of the network. The time complexity of an algorithm is the maximum possible number of time units from the initiation to completion of the algorithm, assuming that the processing delay is negligible and that the maximum delay on an edge is one time

unit. The maximum delay on an edge is the maximum interval, starting from some time when at least one message is being transmitted on the edge, until a message is received on that edge. We consider, however, only algorithms that contain no knowledge of edge delays; ie. algorithms that are event driven with no time outs.

THE SHORTEST PATH PROBLEM

Given a graph (V,E) and given a particular root node, r , find a shortest path tree, using unit edge weights, from r to all other nodes. That is, the path from r to any other given node n in the tree must contain the minimum number of edges over all paths from r to n in the original graph. This problem is often called the minimum hop problem [1], [2], and distributed solutions of this problem form building blocks for many network protocols.

A distributed shortest path algorithm is an algorithm for a communication network to solve the shortest path problem. Initially, the processor at each node is unaware of the network topology but is aware of its adjacent edges, has a copy of the algorithm, and knows whether or not it is the root. The root node starts the algorithm, and the other nodes join in when they receive messages. When the algorithm stops, a shortest path tree has been identified in the sense that each node knows which adjacent edges are in the tree and which one of those adjacent tree edges (called the inedge) is on the path to the root. Each

node also knows its level, ie. the number of edges on the path to the root and the root node, at least, knows that the algorithm is complete. Note that from a communication standpoint, it is not necessary for a node to know the shortest path to the root, since it can send a message on its inedge, and the receiving node can forward it on its inedge, and so forth to the root. Similarly, the root does not have to know the entire tree and can broadcast a message to all nodes by sending it on all adjacent edges. Each node can then forward the message on all tree edges other than the inedge. The nodes on the path to the root from a given node will be referred to as ancestors of the node, and the nodes for which a given node is an ancestor are the descendants of that node.

SIMPLE SHORTEST PATH ALGORITHMS

Note that the shortest path problem with equal edge weights is trivially simple in a synchronous network where all edge delays are the same. The root simply sends a message over each of its edges telling the neighboring nodes that they are neighbors of the root. Each such neighbor sets its level to 1 and sets the edge on which the message was received as the inedge. Each of these nodes immediately sends out a message on all other edges telling the recipients that they are neighbors of a level 1 node. In general, when a node receives a message that it is the neighbor of a level i node, then the node (if not at level i or less itself) sets its level to $i+1$, sets its inedge to

an edge over which the above message was received, and sends a message on the other edges that the recipient is the neighbor of a level $i+1$ node. Because of the uniform delay, each node hears its first message from a node at the next lower level and thus sets its level and inedge correctly. When a given node (other than the root) has heard a message or acknowledgement from each neighbor, it send an acknowledgement on its inedge, informing its immediate ancestor both that that edge is in the tree and that the algorithm is complete for all descendants of the given node. Thus the acknowledgements are collected from the leaves to the root indicating completion of the algorithm. It is easy to see that at most two messages are sent over each edge, one in each direction, yielding a communication complexity of $O(E)$ and a time complexity of $O(V)$ where $E=|E|$ and $V=|V|$.

Note that any shortest path algorithm requires at least one message to be sent on each edge. To see this, suppose that an algorithm failed to send a message over some given edge of some network. Consider modifying the network to contain one or more nodes at the middle of the given edge. These nodes would receive no messages with the given algorithm and thus could not determine their level. Similarly any algorithm requires at least as many time units as the level of the highest level node, which could be $V-1$ in the worst case. Thus any shortest path algorithm requires at least a communication complexity of $O(E)$ and a time complexity of $O(V)$.

For an asynchronous network with unknown edge delays, the above algorithm doesn't work since a node might hear its first

message via a longer path than the shortest path. The classical and most obvious solution to this problem is a distributed form of the Bellman algorithm. This algorithm operates with the same messages as the previous algorithm except that if a node, after setting its level, receives a subsequent message from a node on a shorter path, then that node changes its level and inedge, and sends another message at its reduced level to all other neighbors. This algorithm, generalized to arbitrary edge lengths, was in essence used in the original Arpanet routing algorithm [3]. From a practical point of view, the distributed Bellman algorithm is simple and rarely requires more than a few messages per edge. Since any distributed shortest path algorithm requires at least one message per edge, the search for better algorithms in the communication complexity sense is primarily of theoretical interest.

Another obvious distributed shortest path algorithm, and the basis for our new algorithm, is what we call the basic coordinated shortest path algorithm. This algorithm works in successive iterations, with the beginning of each iteration synchronized by the root node. Initially, the root sends a message to each neighbor indicating the first iteration of the algorithm. Each recipient marks its level as 1, marks the edge on which the message was received as the inedge, and sends an acknowledgement back to the root on the inedge. When the root receives an acknowledgement from each neighbor, it starts iteration 2 by sending an iteration 2 message to each neighbor. Each neighbor then forwards the message to each of its neighbors

other than the root.

In general, after the $i-1^{\text{th}}$ iteration is complete, the Shortest path tree has been formed out to level $i-1$. The root then broadcasts the message for iteration i out on this tree and each node at level less than $i-1$ will rebroadcast the message to its immediate descendants and so forth out to level $i-1$ (see figure 1). The nodes at level $i-1$ rebroadcast the message over all edges other than the inedge (or optionally over all edges not already known to be at lower level than i). Each node, if it first receives a message during iteration i , sets its level at i , sets its inedge to the edge on which the first such message is received, and sends an acknowledgement (ack) on this edge. In addition, for each subsequent message at iteration i or above that the node receives on an edge other than this inedge, the node returns a negative acknowledgement (nak) on that edge, thus informing the sender that the given edge is not in the shortest path tree. Each node at level $i-1$, after receiving an ack or nak to each iteration i message sent, sends an ack on its inedge, including the number of acks that it received. Each node at lower level, after receiving an ack on each outgoing tree edge, sums the numbers accompanying these acks, thus calculating its number of level i descendants. Each non-root node sends an ack with this sum on its inedge. The root node terminates the algorithm if this sum (i.e. the total number of level i nodes) is zero, and otherwise starts iteration $i+1$.

To analyze the algorithm's complexity, note that each broadcast message is accompanied by exactly one ack or nak in the

opposite direction over the same edge. Each node sends at most one ack per iteration. Each node also sends at most one nack over each adjacent edge not in the shortest path tree. Thus, letting L be the number of iterations, the total number of acks is at most $2E+LV$, and the total number of all kinds of messages is at most $4E+2LV$ or $O(V^2)$. We can identify the term $2LV$ as the cost of synchronizing the algorithm. The time required to synchronize the i^{th} iteration is $2i$, and thus the time complexity for synchronization (and thus for the entire algorithm) is $O(L^2) = O(V^2)$. If one considers a simple line network with the root at one end, one sees clearly how inefficient this algorithm is.

PREVIOUS WORK ON DISTRIBUTED SHORTEST PATH ALGORITHMS

Subsequent work on distributed shortest path algorithms has concentrated on reducing the cost of synchronization from that of the coordinated algorithm. One approach, taken by Gallager [1] and Frederickson [4], processes a group of levels at a time, synchronizing back to the root only once for each group. In the algorithm of [4], each group consists of $V/\sqrt{E}$ levels. The distributed Bellman algorithm is used within a group and the basic coordinated algorithm is used between groups. This leads to a communication and time complexity of $O(V\sqrt{E})$, which ranges from $O(V^{1.5})$ for very sparse graphs to $O(V^2)$ for very dense graphs. Another approach, taken by Awerbuch [5], was based on constructing a subgraph on which to perform the synchronization. This led to a communication complexity of $O(V^2)$ and a time

complexity of $O(V \log V)$.

INFORMAL DESCRIPTION OF THE NEW ALGORITHM

The new algorithm to be described is similar to that of [1] and [4] in that it is based on the basic coordinated algorithm, synchronizing only between successive groups of nodes. The difference is that here the number of levels processed in each group is a function of the number of nodes in that group. We shall see that the communication complexity of the resulting algorithm is $O(V^{1.6} + E)$ and the time complexity is $O(V^{1.6})$.

The algorithm exercised by the root node depends on V , which is first computed as follow. First, the root broadcasts a message to all of its neighbors. Each node, upon receiving the message for the first time, sets its inedge to the edge on which the message is first heard and broadcasts the message to all of its other neighbors. The set of inedges so formed constitutes a spanning tree of the graph. Each edge also sends a nak on each edge other than the inedge on which the message is first received. The leaves of the spanning tree (other than the root) are identified by the receipt of naks on each adjacent edge other than the inedge. Each leaf node then sends a report message on its inedge reporting that it is a single node. Each other node, after hearing either a nak or a report from each adjacent edge other than the inedge, adds the number in each report of nodes, and adds one for itself. The non-root nodes report this number back on their inedges, whereas for the root, this number is the

total number of nodes V in the network. The root then broadcasts the number of nodes and edges over the spanning tree, and an acknowledgement is collected back to the root starting at the leaves.

At this point, the root starts the algorithm proper. The nominal number of levels to be processed in each iteration is $\lfloor V^{0.2} \rfloor$, where $\lfloor x \rfloor$ means the integer part of x . The circumstances under which the actual number of levels processed in an iteration differs from this nominal value will be discussed later.

Initially the root uses the basic coordinated algorithm to construct a shortest path tree out to this desired number of levels. At the end of this and each subsequent iteration, a shortest path tree has been constructed out to some given level l . The root then broadcasts a "global" message out to all of the nodes at level l instructing them collectively to extend the shortest path tree over some subsequent group of levels. The nodes at level l are called synch nodes for this new iteration of the algorithm. Each synch node is responsible for coordinating the search for nodes whose shortest path to the root passes through that synch node (ie. for finding its own descendants in the expanded shortest path tree). Essentially each synch node applies the basic coordinated algorithm (regarding itself as the root), exploring the number of levels in the new group. Figure 2 illustrates this process, showing the beginning of the iteration with a global broadcast from the root to all the synch nodes, then showing the individual local cycles of each synch node in exploring subsequent levels, and finally showing the synch nodes

propagating a global acknowledgement, as in the basic coordinated algorithm, back to the root.

There is a complication to the above rather simple structure, due to the fact that there is no coordination between the different synch nodes. Thus the local trees grown by some synch nodes may progress quickly, while others progress slowly. In this case, a node can be seized by a quickly growing local tree, using a path longer than that through some other synch node. A slower tree grown by some other synch node will then eventually reach the given node and provide a shorter path to the root. An important property of the algorithm is that a node moves from one local tree to another only when its path length to the root is decreased. Thus a node can never join the same tree more than once, and the number of times a node can change local trees is upper bounded both by the depth of the local trees and by the number of synch nodes.

Figure 3 illustrates the process of synch nodes growing local trees and of a node first joining one local tree and then another. This figure indicates a new complication to be considered. It is possible for a quickly growing local tree from one synch node to have processed many more levels than some slower tree, and because of this, a node that moves from the quick tree to the slow tree may have many descendants in the fast tree, all of which will have to change trees. Thus many extra messages can be generated by a single change, and several complicating factors must be introduced into the algorithm to limit this message growth.

First, each node, on the cycle after joining a local tree, must send out messages on each adjacent edge. Thus if a node has very high degree and also changes trees often, these adjacent edge messages dominate the communication complexity for the entire algorithm. This problem is cured by freezing the growth of a local tree whenever it seizes a node of very high degree. In a manner to be described next, this causes some global synchronization between the local trees, and the frozen tree remains frozen until it becomes clear that the node of high degree is in fact on the correct local tree. The appropriate threshold of node degree to cause this freezing turns out to be $\sqrt[0.4]{V}$.

Recall that the number of levels that each synch node attempts to process is $\lfloor V^{0.2} \rfloor$. Each synch node, at its own speed, uses the coordinated algorithm to construct its own local shortest path tree, omitting nodes that already have shorter or equally short paths through other synch nodes. This construction ceases when one of the three following conditions occurs: 1) the number of levels processed reaches the attempted depth $\lfloor V^{0.2} \rfloor$; 2) the tree is frozen at some smaller depth by seizing a node of degree greater than $\sqrt[0.4]{V}$; and 3) the tree finds no new nodes at some level before reaching the attempted depth. We refer to the trees and synch nodes for cases 1, 2, and 3 above as active, frozen, and inactive respectively.

Each synch node, after completing its local tree as above, sends a global ack back on its inedge. If there are frozen

trees, however, a shortest path tree is not fully constructed out to the attempted depth of the new group. In this case, we regard the just completed process of designating the synch nodes, constructing the local trees as above, and reporting back to the root as a sub-iteration. The root determines (from the information reported back) the smallest level at which some tree became frozen. We are assured that the shortest path tree is completely known out to that level, and therefore the tree or trees frozen at that level can be unfrozen. The root then starts a new sub-iteration, using the same synch nodes as before and restarting the growth of those trees that had been frozen at that lowest level. A synch node that is frozen at some higher level checks to see if it is still frozen (ie. the high degree node might have been seized by another tree). If so, it immediately sends an ack back to its inedge, and if not, it continues growing its tree. Each synch node that is unfrozen in this new sub-iteration continues to grow its tree until again one of the three conditions above occurs.

After some number of sub-iterations, the root will find that there are no more frozen trees and that the global shortest path tree has been extended by the attempted $\lfloor V^{0.2} \rfloor$ levels. At this point, the final complicating factor in the algorithm occurs. If the final shortest path tree has a large number of levels with few nodes at each level, then the global synchronizing required between iterations will still dominate the communication complexity. On the other hand, in this situation, there will be relatively few synchronizing nodes at each

iteration, and therefore relatively few opportunities for nodes to change levels. In this case, it is advantageous to increase the number of levels in a group to $\lceil V^{0.4} \rceil$. We describe the conditions for this increase more precisely after describing more of the details of the algorithm.

DETAILED DESCRIPTION AND COMPLEXITY ANALYSIS

The actual algorithm is contained in the appendix. In this section we give some of the implementation details and describe some of the properties of the algorithm pursuant to calculating its communication complexity. There are only four types of messages used in the algorithm proper (ie. after the calculation of V) - global broadcasts, global acks, local broadcasts, and local acks. The global broadcasts are used at the beginning of each sub-iteration and are propagated from the root out on the currently formed part of the shortest path tree. These broadcast messages normally carry three parameters--the level of the synch nodes, and the initial and desired final levels for the sub-iteration. When a synch node has completed its task (by becoming frozen, inactive, or processing out to the desired level), it sends a global ack back on its inedge, and when a node at a lower level receives global acks from each of its immediate descendants, it forwards the global ack on its inedge. Each global ack contains two parameters. The first is the minimum

level at which freezing occurred, minimized over the synch nodes that are descendants of the node sending the global ack, and the second is the number of descendant synch nodes that are active or frozen.

When the root receives these global acks from each neighbor, the root knows whether freezing occurred and the minimum level of such freezing. If freezing occurred at some level lower than the desired level of the sub-iteration, then the root starts another sub-iteration with the same synch nodes and with a starting level equal to the minimum freezing level plus 1 (since the shortest path tree is correctly constructed out to the minimum freezing level). If the sub-iteration has completed the shortest path tree out to the desired level of $\lfloor V^{0.2} \rfloor$ beyond the synch nodes, then the root node starts a special sub-iteration, using the same synch nodes and covering only a single level, one beyond the previous level. If the number of active synchs at the end of this special sub-iteration exceeds $\lfloor V^{0.2} \rfloor$, then the iteration ends and a new iteration starts with the nodes at this newly calculated level as synch nodes. Otherwise, another sub-iteration starts, using the old synch nodes and using a desired level $\lfloor V^{0.4} \rfloor$ beyond that of the synch nodes.

The reason for this rather peculiar sub-iteration with a single level is that no node can be seized from one local tree to another during this sub-iteration. Thus a local tree reported as active actually contains at least one node at the new level and a path back to the synch node. Thus it contains at least $V^{0.2}$ nodes. If the number of active synch nodes exceeds $\lfloor V^{0.2} \rfloor$, then

at least $V^{0.4}$ new nodes have been added to the shortest path tree in the entire iteration. Note that for the earlier sub-iterations, active synch nodes might lose most of the nodes in their local trees when other slower trees seize those nodes away.

When an iteration is extended to cover $[V^{0.4}]$ nodes, then, if any synch nodes are active at the end of the iteration, the iteration must add at least $V^{0.4}$ nodes to the shortest path tree; if no synch nodes are active (or frozen) at the end of any sub-iteration, then, of course the algorithm is finished and the shortest path tree is complete. Thus we have shown that at least $V^{0.4}$ nodes are added to the shortest path tree in every iteration except perhaps the final iteration. Thus the algorithm contains at most $V^{0.6}+1$ iterations.

Observe that each node can receive at most one global broadcast message per sub-iteration. Thus we can bound the total number of global broadcast messages by bounding the number of sub-iterations. Suppose first that there are F sub-iterations caused by frozen trees. Each such sub-iteration unfreezes at least one node of degree greater than $V^{0.4}$, and nodes unfrozen in different sub-iterations belong to different levels of the final shortest path tree. Thus there are at least F levels in the final tree containing nodes of degree greater than $V^{0.4}$, and we select one such node in each of these levels. Since a neighbor of a node at level l must be at level l or an adjoining level, we see that a node can be a neighbor of at most 3 selected nodes. Thus $3V > FV^{0.4}$, and thus

$$F \leq 3V^{0.6}$$

(1)

In addition to the sub-iterations caused by frozen trees, each iteration starts with a sub-iteration of $\lfloor V^{0.2} \rfloor$ desired levels, then performs the special sub-iteration with one level, and finally, perhaps, performs the final sub-iteration with a final desired level $\lfloor V^{0.4} \rfloor$ beyond the synch node. Thus there are at most three sub-iterations per iteration beyond those caused by freezing. There are at most $V^{0.6} + 1$ iterations, leading to at most $3(V^{0.6} + 1) + 3V^{0.6}$ sub-iterations altogether. There are no global broadcast messages on the initial iteration and at most V for each subsequent sub-iteration. Thus the total number G of global broadcast messages satisfies

$$G \leq 6V^{1.6}$$

(2)

Next consider the local broadcasts and acks. We first show that a node can join at most $\lfloor V^{0.2} \rfloor$ local trees. If a node first joins a local tree when the desired final level is $\lfloor V^{0.2} \rfloor$ beyond the synch level, this result follows since each change of local tree is accompanied by a decrease in level. If a node first joins a local tree when the desired final level is $\lfloor V^{0.4} \rfloor$ beyond the synch level, this result follows since the number of active synch nodes is at most $\lfloor V^{0.2} \rfloor$.

Consider now what the algorithm does when a node changes levels. During each cycle of the algorithm for growing a local tree, a node can be in one of two phases. In the first phase, it

has not yet received a local broadcast message, and in the second phase, it has passed on the local broadcast and is waiting for acknowledgements from each active edge. The variable count is used in the algorithm to keep track of how many acknowledgements are awaited, and a value of zero for count indicates the first phase. When a local broadcast message arrives at a node from a different tree, the cycle number is compared with the node's current level, and if smaller, the node changes trees. At this point, if the count variable is positive, the node returns an acknowledgement on the old inedge with unused state, indicating to the old parent node that the node has moved to another tree. Alternatively, if the count is zero, the node sends nothing over the old inedge until the next local broadcast message is received, at which time it returns an ack with unused state. Thus, precisely one ack is sent for each local broadcast, even when nodes change trees.

In counting the number of local broadcast messages for the algorithm, we consider two types of terms separately, first the initial local broadcasts sent by a node after changing level, and second all the other local broadcasts. For the first type, we note that as a result of the freezing, each node of degree more than $v^{0.4}$ sends at most one such message on each outgoing edge, and each node of smaller degree sends at most $v^{0.2}$ messages on each of its at most $v^{0.4}$ outgoing edges. Thus the total number of local broadcast messages of this first type, L_1 , is at most

$$L_1 \leq 2E + v^{1.6} \quad (3)$$

In counting the second type of local broadcast messages, we look at the number of such messages that can be received by each node. Such messages are only sent on edges for which the previous local broadcast was acknowledged with an active or frozen state, so they can only be received on the inedge of a node (or what was the inedge when the previous ack was sent). While the node belongs to one tree, successive messages on the inedge contain successively higher cycle numbers except in the special case where a new sub-iteration is started because of freezing. In that special case, synch nodes frozen at higher than the minimum level repeat the previous cycle number to test if the local tree is still frozen. Since the nodes that receive these special local broadcast messages do not receive global broadcast messages for the given sub-iteration, we can regard these special local broadcast messages as included in our previous bound on global broadcasts. Ignoring these special messages, a node can receive at most $\lceil v^{0.4} \rceil - 1$ local broadcast messages of this type for each local tree that it joins. Since there are at most $v^{0.2}$ such trees, the total number of local broadcast messages of second type is

$$L_2 \leq v^{1.6} \quad (4)$$

Combining (2), (3), (4) and recalling that there is one ack message for each broadcast message, the total number of messages C for the algorithm is

$$C \leq 12V^{1.6} + 4E \quad (5)$$

Thus $C = O(E+V^{1.6})$ as claimed.

To bound the time T required by the algorithm, note first that $T \leq C$ since the algorithm is event driven and at least one message must be sent each time unit. Note also that the term $4E$ in (5) came from the initial local broadcasts from frozen nodes and their acknowledgements. Since a node is assumed to send its local broadcasts in parallel, and the acknowledgement for each is returned with no waiting, we see that the term E can be replaced with V for the time bound, and $T = O(V^{1.6})$.

APPENDIX: THE ALGORITHM

Local Variables at nodes

All variables are integer values except where noted otherwise. The initial values of all of the variables are arbitrary, except for the variable *level*, whose initial value is ∞

cycle = current level of the exploration in the local algorithm.

count = counter of unacknowledged LOCAL_BROADCAST messages.

desired_level = the last level to be processed in the iteration

initial_level = the first level to be processed in the sub-iteration.

level = current estimate of the node's level in shortest path tree.

status = estimate of whether node has frozen descendents, active descendents, or only inactive descendents; only meaningful for nodes in local trees. Possible values are {Active,Frozen,Inactive}.

state(e) = estimate of whether there are frozen descendents, active descendents, or only inactive descendents on the further side of tree edge *e*. If *e* is not a tree edge, *state(e)*=unused. It is kept for each incident edge *e*. Possible values are {Frozen,Active,Inward,Inactive,Unused}.

synch_level = the level of synch nodes.

synchs_number =

estimate of the number of active synchs which are descendents of that node; is meaningful only for nodes in the global tree.

Messages sent during the algorithm

GLOBAL_BROADCAST(*s,i,d*) =

message by which the root triggers execution of the next sub-iteration. The message carries the following parameters: *synch_level*, *initial_level*, *desired_level*.

GLOBAL_ACK(*c, m*) =

message by which a node reports on its inedge about the termination of the sub-iteration. The message carries the following parameters: cycle, synchs_number.

LOCAL_BROADCAST(*c*) =

message by which a synch node triggers the local exploration process. The parameter *c* is the cycle number.

LOCAL_ACK(*c, s*) =

message by which a node reports about the termination of the local exploration process. The parameters of the message are *c* (the cycle number) and *s*. The latter determines the state of the edge.

Procedures used in the algorithm:

XMIT LOCAL_BROADCAST =

generates *LOCAL_BROADCAST* messages.

XMIT GLOBAL_BROADCAST =

generates *GLOBAL_BROADCAST* messages.

SYNCH_CONTROL =

procedure executed by the synch nodes after termination of each cycle.

ROOT_CONTROL =

procedure executed by the root node after termination of each sub-iteration.

Initialization of the algorithm (by the root node)

begin

if the set of incident edges is empty then STOP /* the algorithm is complete */

else

begin

cycle := 1;

level := 0;

synch_level := 0;

desired_level := $\lceil \nu^{0.4} \rceil$;

initial_level := 1;

for all edges e , state(e) := active;

Call Procedure XMIT_LOCAL_BROADCAST;

end

** of the initialization */*

ode algorithm

Procedure XMIT_LOCAL_BROADCAST

count := 0;

status := inactive;

for each edge e such that state(e) = active or frozen

begin

count := count + 1 ;

send LOCAL_BROADCAST(cycle) on edge e ;

end

count = 0 then

begin

status := inactive;

send LOCAL_ACK(cycle, inactive) on inedge;

end

** of the procedure */*

use to receipt of LOCAL_BROADCAST(p) on edge e :

```

p < level then /* node joins new tree at lower level */
begin
  if count > 0 and level  $\neq \infty$  then send ACK(cycle,unused) on inedge;
  /* terminating branch of old tree if ack awaited */
  level := p;
  cycle := p;
  inedge := e;
  state(e) := inwards;
  synch_level := 0; /* initial value, to ensure that level > synch_level */
  for all edges e'  $\neq$  e, state(e') := active;
  if node_degree >  $V^{0.4}$  then status := Frozen
  else status := active;
  send LOCAL_ACK(cycle,status) on inedge;
end
else /* p  $\geq$  level */
  if e = inedge then
    if p > level then /* broadcast must be propagated outward */
      begin
        cycle := p;
        Call Procedure XMIT_LOCAL BROADCAST;
      end
    else send LOCAL_ACK(cycle,status) on inedge;
    /* in this case p = level; this means that this is the check for frozen nodes
    and the current node is in the last layer */
  else /* e  $\neq$  inedge */
    begin
      send LOCAL_ACK(p,unused) on edge e;
      state(e) := unused;
    end
  /* this occurs for p  $\geq$  level, e  $\neq$  inedge and negatively acknowledges a broadcast

```

when old tree is retained */

of the response */

on receipt of LOCAL_ACK(p,s) on edge e

p = cycle then

outdated acks satisfy $p > \text{cycle}$; they are ignored */

begin

if s=frozen then status :=frozen;

else if s=active and status≠frozen then status :=active;

state(e) :=s;

count :=count -1;

if count=0 then

if level > synch_level then /* the node is not a synch */

send LOCAL_ACK(cycle, status) on inedge;

else Call Procedure SYNCH_CONTROL

end

/* of the response */

procedure SYNCH_CONTROL

level=0 and status = frozen then status :=active;

the root ignores freezing on its local iteration */

status = active and cycle < desired_level then

begin

cycle :=cycle +1;

Call Procedure XMIT_LOCAL_BROADCAST

end

e /* local exploration process has been freezed or terminated */

begin

```

if status =inactive then synchs_number :=0;
else synchs_number :=1;
    if level > 0 then
        send GLOBAL_ACK(cycle,synchs_number) on inedge
    else if level=0 then
        /* it is the end of the local cycle at the the root */
        Call Procedure ROOT_CONTROL;

```

```

end

```

```

end /* of the procedure */

```

Procedure XMIT_GLOBAL_BROADCAST

```

begin

```

```

    count :=0;

```

```

    synchs_number :=0; /* lower bound */

```

```

    cycle :=  $\infty$ ; /* upper bound */

```

```

    for each edge  $e$  such that state( $e$ ) =active or frozen

```

```

        begin

```

```

            count :=count +1 ;

```

```

            send GLOBAL_BROADCAST(synch_level,initial_level, desired_level) on  $e$ ;

```

```

        end

```

```

end /* of the procedure */

```

Response to receipt of GLOBAL_BROADCAST(s,i,d) on edge e :

```

begin

```

```

    if there exists no incident edge  $e$  with state( $e$ )=active or frozen then

```

```

        send GLOBAL_ACK(cycle,0) on inedge;

```

```

    else

```

```

        begin

```

```

            synch_level :=  $s$ ;

```

```

            initial_level :=  $i$ ;

```

```

desired_level := d;
if level < synch_level then Call Procedure XMIT_GLOBAL_BROADCAST
else /* level = synch_level */
    begin
        if initial_level = cycle + 1 then cycle := cycle + 1;
        /* otherwise test for still frozen local tree */
        Call Procedure XMIT_LOCAL_BROADCAST;
        /* this invokes execution of the local exploration process */
    end
end
end /* of the response */

```

Response to receipt of GLOBAL_ACK(c,m) on edge e :

```

begin
    synchs_number := synchs_number + m;
    if m > 0 then
        begin
            cycle := min {cycle, c};
            state(e) := active;
        end
    else state(c) := inactive;
    count := count - 1;
    if count = 0 then
        if level > 0 then send GLOBAL_ACK(cycle, synchs_number) on inedge
        else Call Procedure ROOT_CONTROL;
    end /* of the response */

```

Procedure ROOT_CONTROL

```

begin
    if synchs_number = 0 then STOP ; /* algorithm is complete; the tree is given by

```

*inedges pointing in and inactive edges pointing out */*

else

begin */* next sub-iteration */*

initial_level := cycle + 1 ; */* starting at the height of lowest tree */*

if cycle = desired_level then */* otherwise freezing exists and synch_level
and desired_level need not be changed*/*

if synchs_number < $V^{0.2}$ and desired_level - synch_level < $\lceil V^{0.4} \rceil$ then

desired_level := synch_level + $\lceil V^{0.4} \rceil$;

/ few synchs, make group long */*

else

if desired_level = synch_level + $\lceil V^{0.2} \rceil$ then

desired_level := desired_level + 1 ;

/ the estimate of active synchs may be too high; test actual
number of active synch nodes by processing just one layer */*

else

begin */* new iteration */*

synch_level := cycle;

desired_level := synch_level + $\lceil V^{0.2} \rceil$;

end

Call Procedure XMIT_GLOBAL_BROADCAST

end

end */* of the Procedure */*

REFERENCES

- [1] R. G. Gallager, "Distributed Minimum Hop Algorithms", M.I.T. Technical Report, LIDS-P-1175, January 1982.
- [2] A. Segall, "Distributed Network Protocols", IEEE Trans. I.T., Jan. 1983, pp. 23-35.
- [3] F. E. Heart, S. M. OrNSTein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network", 1970 Spring Joint Computer Conference, AFIPS Conference Proceedings.
- [4] G. Frederickson, "A Single Source Shortest Path Algorithm for a Planar Distributed Network", Proc. 2nd Symp. on Theoretical Aspects of Computer Science, Jan. 1985.
- [5] B. Awerbuch, "An Efficient Network Synchronoization Protocol", ACM Syposium on Theory of Computing, April 1984, Washington.

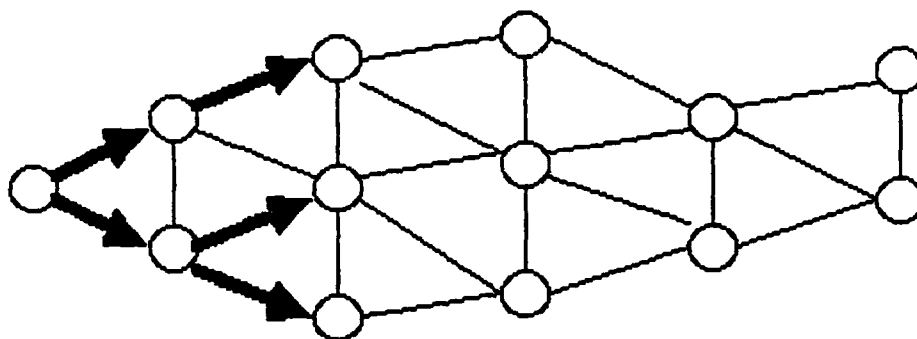


FIGURE 1A BROADCAST TO FIND A NEW LEVEL

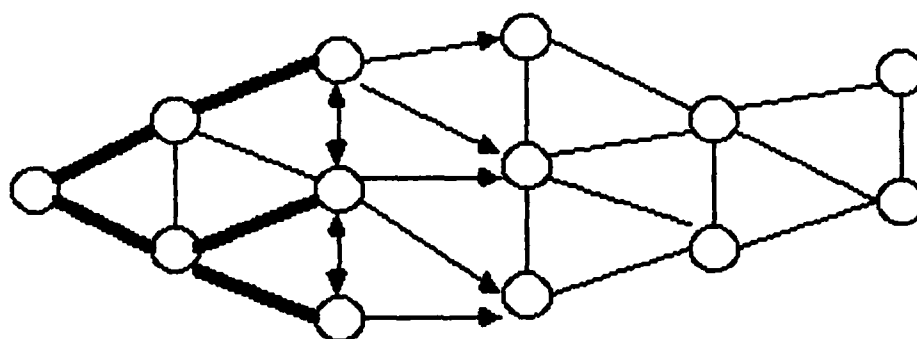


FIGURE 1B TESTING NODES AT NEW LEVEL

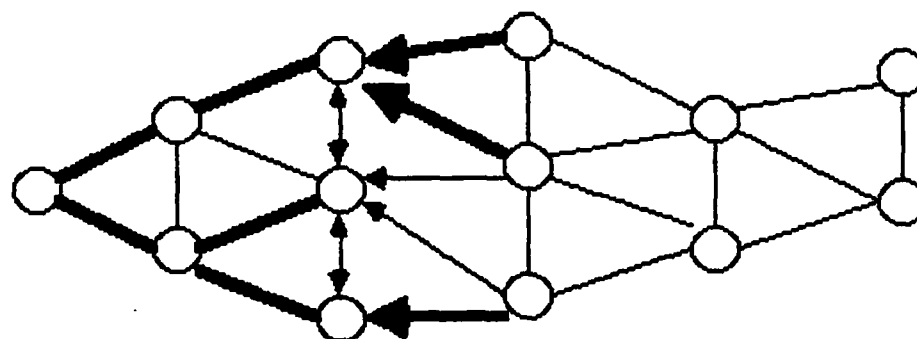


FIGURE 1C ACKNOWLEDGEMENTS

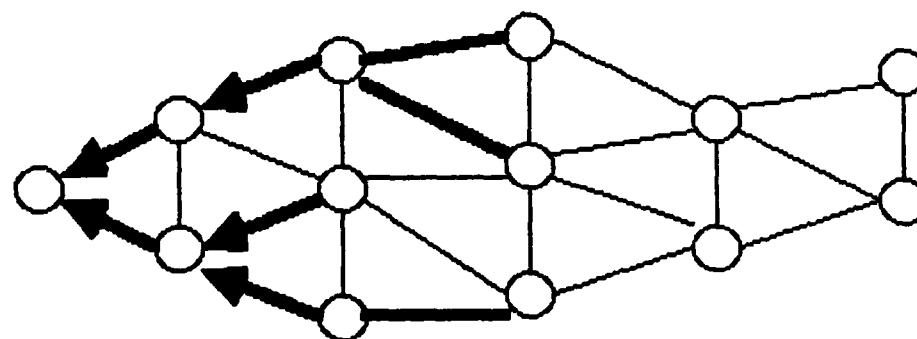


FIGURE 1D COORDINATION BEFORE NEXT LAYER

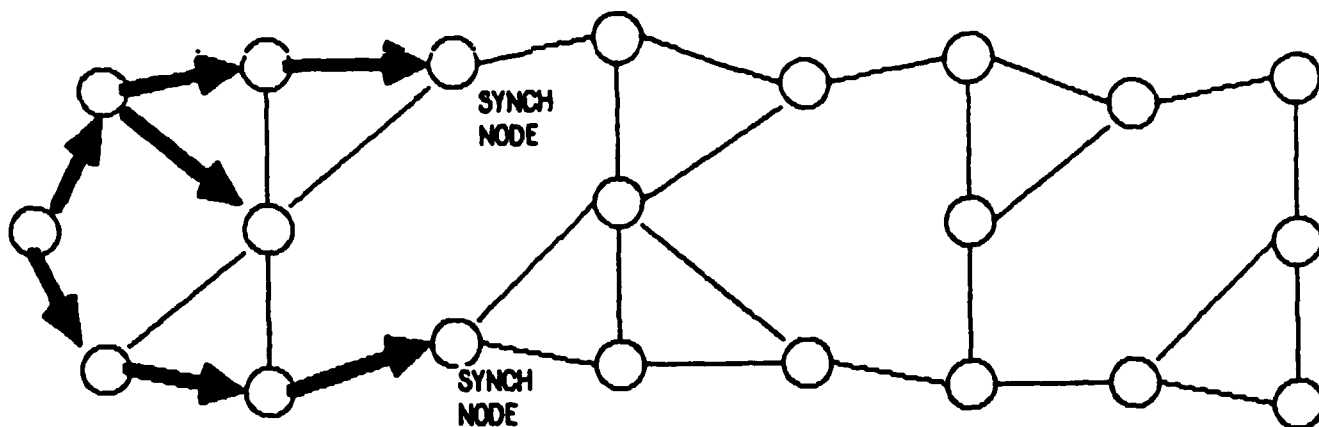


FIGURE 2A: GLOBAL BROADCAST

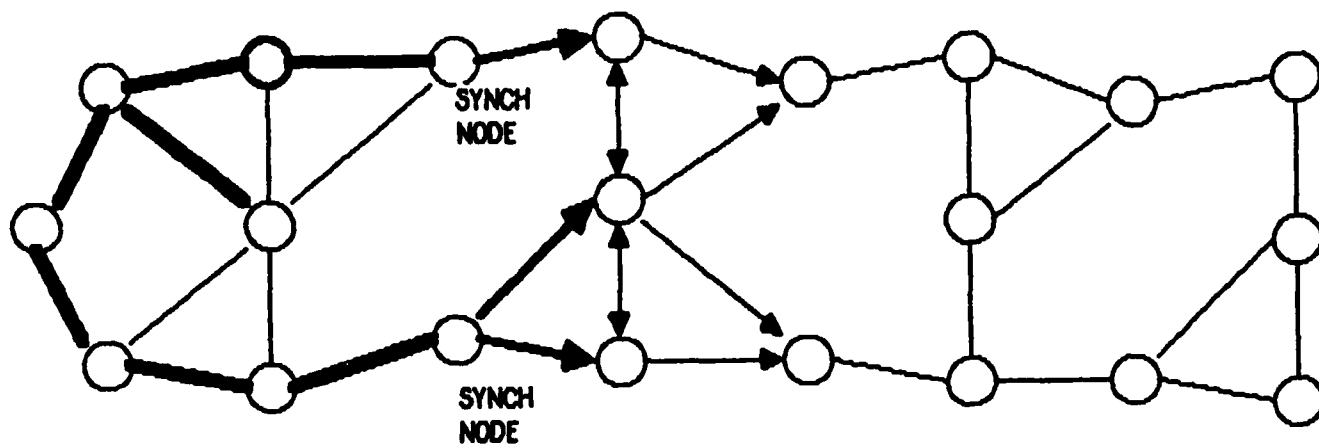


FIGURE 2B: LOCAL BROADCAST

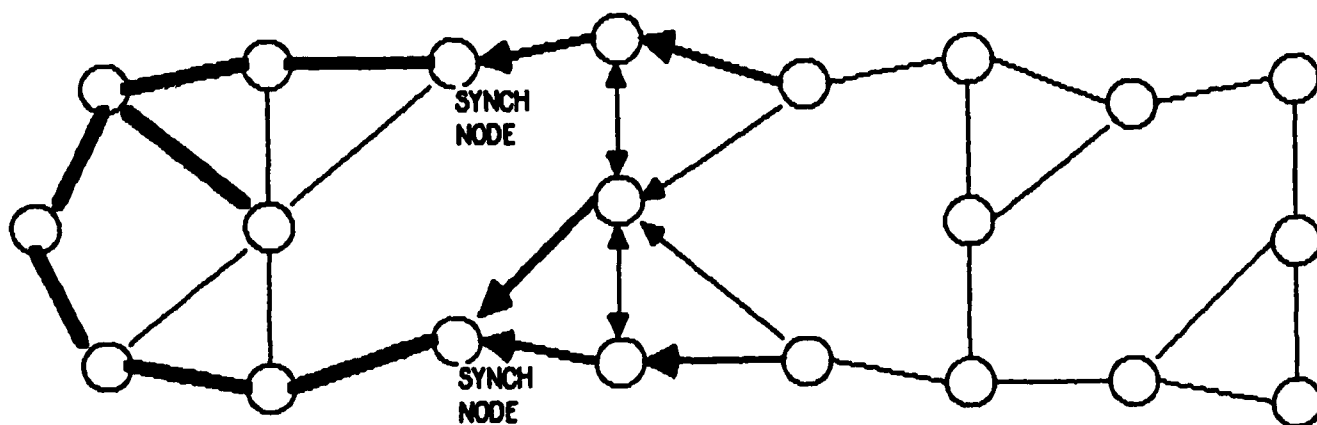


FIGURE 2C: LOCAL ACKNOWLEDGEMENT

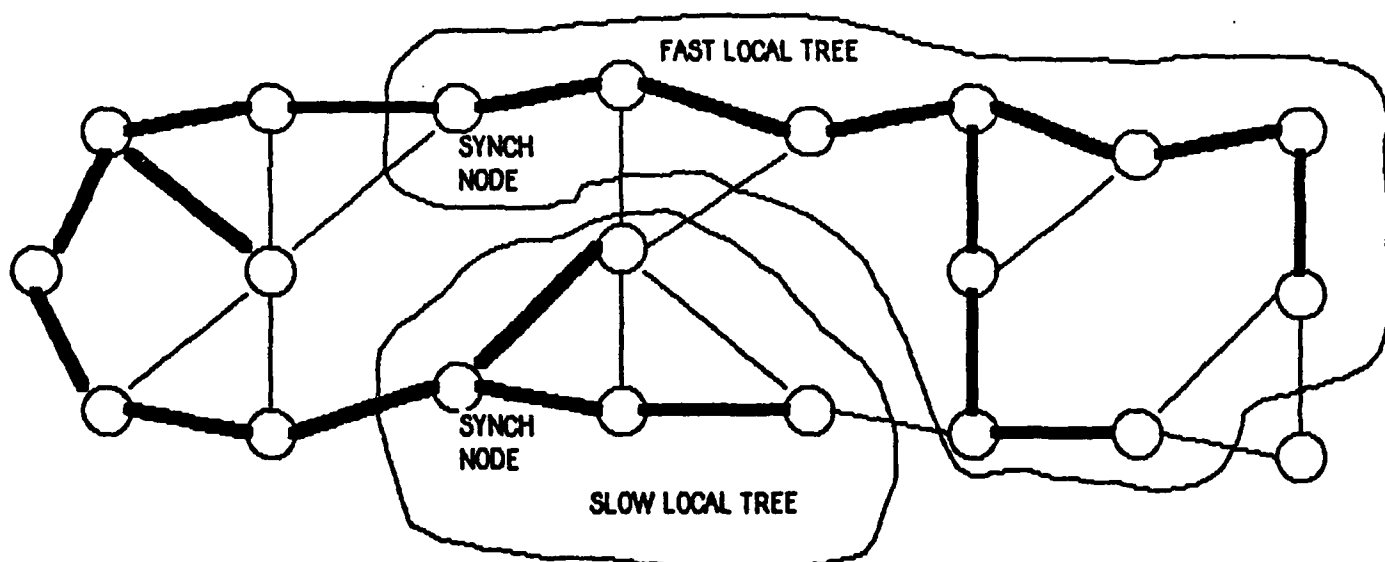


FIGURE 3A: EXAMPLE OF FAST TREE TAKING EXTRA NODES

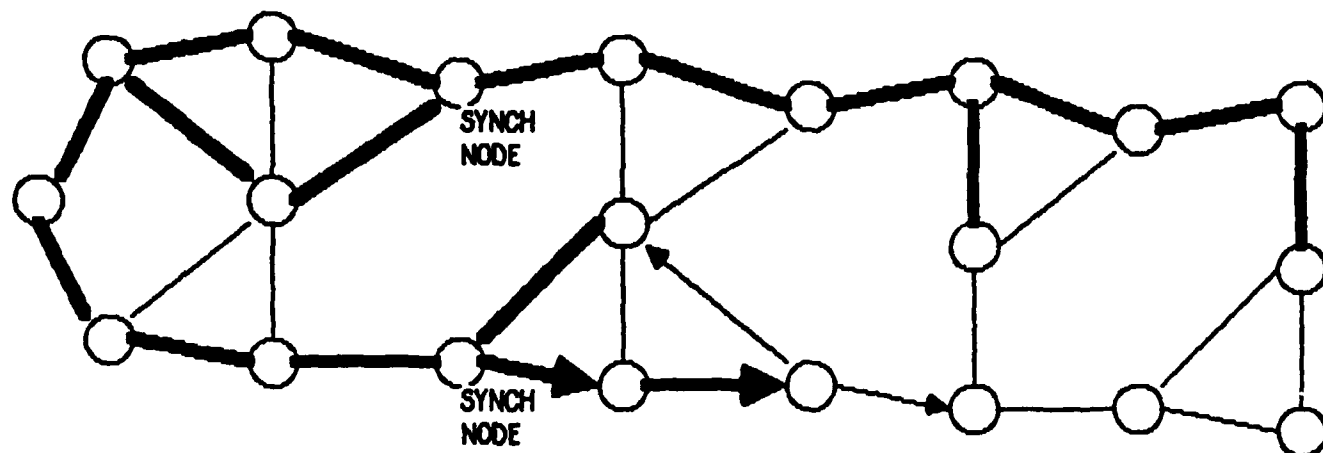


FIGURE 3B: LOCAL BROADCAST IN SLOW TREE RECAPTURING NODE

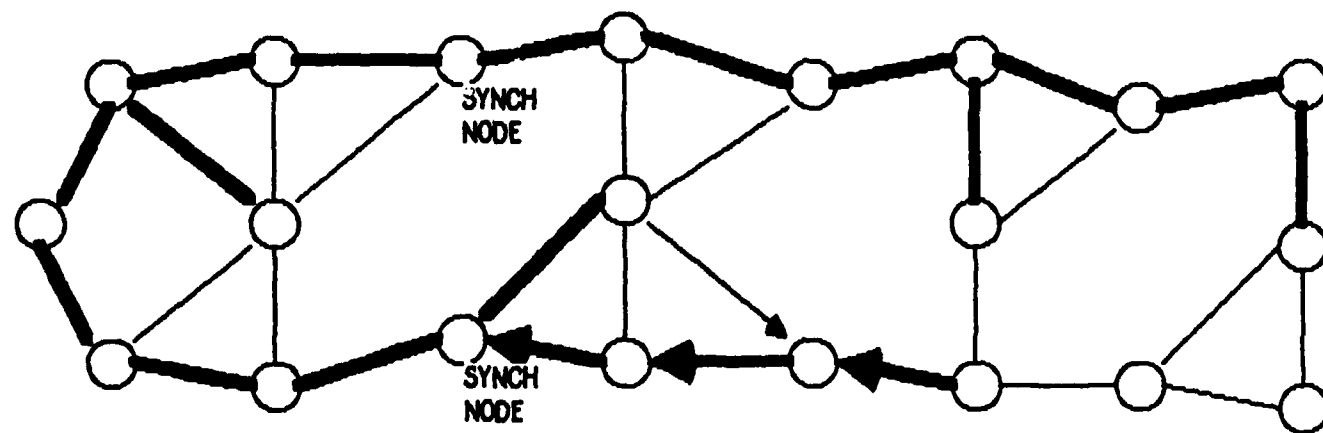


FIGURE 3C: LOCAL ACKNOWLEDGEMENT IN SLOW TREE

Distribution List

Defense Documentation Center 12 Copies
Cameron Station
Alexandria, Virginia 22314

Assistant Chief for Technology 1 Copy
Office of Naval Research, Code 200
Arlington, Virginia 22217

Office of Naval Research 2 Copies
Information Systems Program
Code 437
Arlington, Virginia 22217

Office of Naval Research 1 Copy
Branch Office, Boston
495 Summer Street
Boston, Massachusetts 02210

Office of Naval Research 1 Copy
Branch Office, Chicago
536 South Clark Street
Chicago, Illinois 60605

Office of Naval Research 1 Copy
Branch Office, Pasadena
1030 East Greet Street
Pasadena, California 91106

Naval Research Laboratory 6 Copies
Technical Information Division, Code 2627
Washington, D.C. 20375

Dr. A. L. Slafkosky 1 Copy
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D.C. 20380

Office of Naval Research
Code 455
Arlington, Virginia 22217

1 Copy

Office of Naval Research
Code 458
Arlington, Virginia 22217

1 Copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, California 92152

1 Copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, Maryland 20084

1 Copy

Captain Grace M. Hopper
Naval Data Automation Command
Code OOH
Washington Navy Yard
Washington, DC 20374

1 Copy

Advanced Research Projects Agency
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209

1 Copy

Dr. Stuart L. Brodsky
Office of Naval Research
Code 432
Arlington, Virginia 22217

1 Copy

Prof. Fouad A. Tobagi
Computer Systems Laboratory
Stanford Electronics Laboratories
Department of Electrical Engineering
Stanford University
Stanford, CA 94305