The Design of a Topology Information Maintenance Scheme for a Distributed Computer Network

William D. Tajibnapis
The University of Michigan
MERIT Computer Network

Abstract

In order for the nodes of a distributed computer network to communicate, each node must have information about the network's topology. Since nodes and links sometimes crash, a scheme is needed to update this information. One of the major constraints on such a scheme is that it may not involve a central controller.

In this report a straightforward scheme involving adjacency matrices and a broadcast scheme are discussed and their inadequacies described. The NETCHANGE Protocol which is based on Baran's "Hot Potato Heuristic Routing Doctrine," is presented. This system has been implemented on the MERIT Computer Network and its correctness has been proved.

We end by showing how the NETCHANGE Protocol can be modified into a shortest path algorithm.

Keywords: Distributed Computer Network, Distributed Control, Routing Problem in Networks, Store-and-Forward Message Switching.

Introduction

This paper will discuss the problem of maintaining up-to-date information in the nodes of a distributed computer network. Since such a network has no central node that controls the flow of messages, all nodes are equals. The nodes are interconnected by telecommunication lines. It is a matter of indifference to our discussion whether the nodes are minicomputers connected to one or more "host" computers (which are the computers of interest to the users) or whether the nodes are the host computers themselves. Examples of distributed computer networks are the MERIT Computer Network and the ARPANET.

The Problem

In a network of any size at all, it quickly becomes impractical to have a direct connection between every two nodes. In order for the network to route messages between nodes A and B, it may be necessary to route these messages via nodes C, D and E. Thus, it is essential for every node (i.e., minicomputer) in the network to have some knowledge of the topology of the network.

It is a simple matter to store and use such topology information, but no real computer network is ever static, especially a large one. Telephone lines sometimes fail; computers sometimes crash. Thus a scheme or protocol is needed to keep the topology information up-to-date. The only practical way to do this is to send messages carrying information about topology changes along the very network that is experiencing these changes.

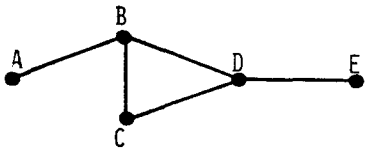
Solutions that involve a central controller of some sort violate the distributed nature of the network and will not be discussed here. (There also is a problem in recovering from a crash in the central controller.) At least one computer network (TYMNET) uses such a scheme, proving that it is workable.

We will discuss several solutions to the topology-information maintenance problem in distributed networks. In doing so we hope to illuminate some of the problems that arise in a system that has distributed control, and which therefore also has a distributed information base.

A Straightforward Solution

The topology of a network can be stored within the node computers in a compact manner in an adjacency matrix (Fig. 1). An entry (i,j) of the matrix is 1 if there is a link between nodes i and j; otherwise it is Ø. It is possible to determine whether there is a path of length k between any two nodes by doing at most k matrix multiplications. Since the maximum length of a path without loops in a network of N nodes is N-1, it is possible to determine whether there is any path at all between two nodes by

doing at most N-1 matrix multiplications. The rules for adding and multiplying the entries are the standard ones, except that 1+1 = 1.



	A	B	C	D	E
A	0	1	0	0	0
B	1	0	1	0	0
C	0	1	0	1	0
D	0	1	1	0	1
E	0	0	0	1	0

Fig. 1 A Network and its Adjacency Matrix

Thus, if a copy of the adjacency matrix is stored in each node, each node will have complete information about the network topology. The following straightforward scheme then suggests itself for transmitting information about topology changes:

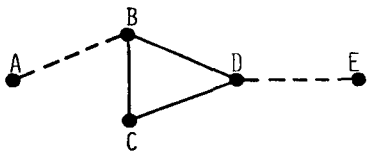
1. When the link ij goes up (comes down), nodes i and j make the appropriate changes to their adjacency matrices. They then send the message 'ij,1' ('ij,0') to every node that they can reach. This message indicates that link ij is now up (down). Whether or not a node can be reached is determined from the updated adjacency matrix.

2. When a node goes down, the situation is treated as if all the links to that node went down.

3. When a node comes up, it obtains a copy of the adjacency matrix from the first (neighboring) node that it establishes contact with.

The above scheme will work as long as all the messages caused by one topology change reach their destinations before the next topology change occurs. However, if several changes occur in quick succession, the scheme may fail miserably, as the following example illustrates.

Consider the network of Fig. 2, in which links AB and DE come up simultaneously. At A's request, B will send A a copy of its adjacency matrix; but note that this matrix will indicate that E is not in the network. Node B will send the message 'AB,1' to nodes C and D, but not to E, since B thinks that E cannot be reached. Similarly E will get a copy of D's matrix, which indicates that A cannot be reached, and D will send the message 'DE,1' to nodes B and C but not to A. Thus when the activity has ceased, A and E will not know that a path exists between them.



	A	B	C	D	E
A	0	0	0	0	0
B	0	0	1	1	0
C	0	1	0	1	0
D	0	1	1	0	0
E	0	0	0	0	0

Fig. 2 Failure of Straightforward Scheme

The flaw of our "straightforward" scheme is now apparent; a sender node (B or D in our example) assumes that its adjacency matrix is correct when it sends out messages about topology changes - that assumption cannot safely be made.

The Broadcast Scheme

Removing the assumption of a valid adjacency matrix implies that the senders must use a broadcast scheme for the topology-change messages. Broadcasting works as follows. The initial sender sends the message to all its neighbors. These nodes in turn send the message to all their neighbors, which in turn send it to all their neighbors, etc. To prevent the message from reverberating around the network forever, it is necessary for each node to keep a list of received messages. When, and if, a message is received a second time, it will already be on the list and is then not transmitted further.

Consider again the example of Fig. 2. When B receives the broadcast message 'DE,1', it will make the change in its adjacency matrix and send the message on to A if the link AB is already up. If AB is not yet up, then when it does come up A will get the up-to-date adjacency matrix and thus will still find out about link DE.

It can, in fact, be proven that, under appropriate conditions, a broadcasted message will eventually be received by all nodes and then will stop reverberating around the network. However, there are still serious problems with the broadcast scheme. The first problem is the list of received messages that each node must keep; this list will get arbitrarily long! One can get around this by keeping only a finite number of messages, and throwing away the oldest message when a new message comes in, under the assumption that the oldest message is no longer roaming around the network.

The second problem is more serious. If the link ij goes down and comes up again shortly thereafter, there is no guarantee that the messages 'ij,0' and 'ij,1', which will be sent out in that order by nodes i and j, will arrive in that same order at every node. If the messages are received in the wrong order,

the node will incorrectly conclude that link ij is down. This problem can only be solved by having sender nodes sequence their messages, but this means that the sequence number has to be preserved across node crashes. This is a tricky business, since most network minicomputers do not have any permanent storage, such as disk.

The third problem is also tricky. If a network breaks into two (or more) disjoint parts, the adjacency matrices will correctly reflect the change, but the matrices of the nodes in part A (Fig. 3) will still contain information about the topology of part B. If anything happens to part B before it is rejoined with A, the matrices in A will not reflect those changes, and problems will arise when the parts are rejoined. Again, schemes can be devised to get around this problem, but they are quite elaborate and clumsy. We are running into a case of diminishing returns -- a rare event is requiring a disproportionate amount of code and effort.

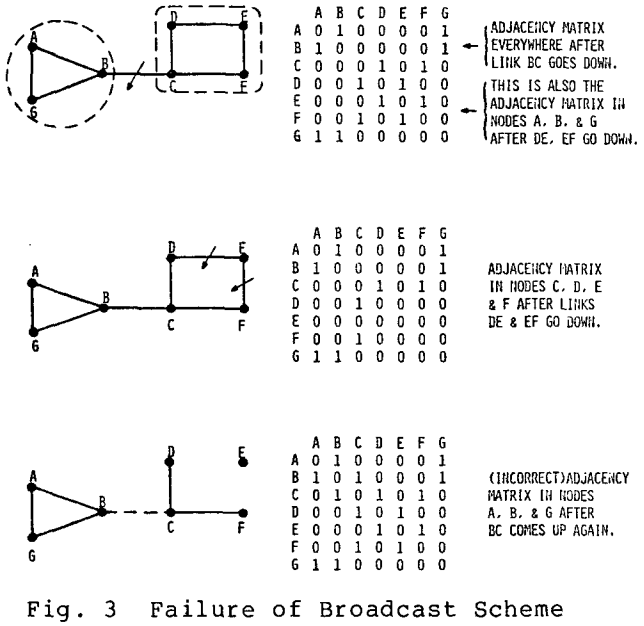


Fig. 3 Failure of Broadcast Scheme

Before we go on to the next (and final) solution, we would like to review the two problems that must be solved in any topology information maintenance scheme.

The first problem is caused by the fact that the messages carrying information about topology changes must travel through the network which is experiencing these changes, and by the fact that there is a time lapse between the occurrence of a change and the arrival of the news of that change at any particular node. Thus a node nevers knows for certain that the topology information that it has is correct.

The second problem is similar; the topology messages need to be sequenced. Since it is virtually impossible for a node to "remember" a sequence number "across" a crash, the sequence number is reset to zero every time a node restarts. This means that a node must resynchronize with the other nodes every time it restarts. This can only be done when the nodes in question know they can communicate, a fact which they learn from the topology messages that need to be sequenced in the first place.

The NETCHANGE Protocol

This next solution solves the two problems just described. The NETCHANGE Protocol is the topology-information maintenance scheme implemented on the MERIT Computer Network; it is based on Baran's "Hot-Potato Heuristic Routing Doctrine".

Now, no matter how rapidly the network is changing, there is one part of the network topology that a node always knows about; the identity of its neighbors (i and j are neighbors if the link ij is up). Furthermore, if and i and j are neighbors, then every time j crashes or restarts, i finds out about it immediately. These facts are a result of the "hand-shaking" protocol that neighboring nodes carry out every time they (re)-establish communication.

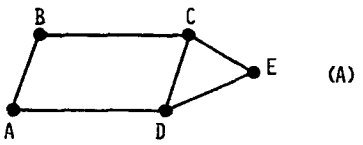
In the NETCHANGE Protocol, the messages carrying information about topology changes (hereafter referred to as NETCHANGE messages) are only sent from one neighbor to another. Since a node always knows who its neighbors are, this solves problem 1. A node always finds out immediately when one of its neighbors restarts; thus two neighboring nodes can always maintain message sequencing between them. This solves problem 2.

The NETCHANGE Protocol differs from the adjacency matrix scheme in the extent of the information stored at each node. In order to relay (or send) a record correctly, a node need not know the complete, exact topology of the network. All it needs to know is in what direction to send the record, i.e., on what link it should transmit the record. In other words, it should know the identity of the first node along the route between itself and the destination node. If there is more than one route, it needs to know the relative lengths of these routes (in our scheme the nodes know the absolute length of these routes).

For example, consider node A in the network of Fig. 4a. The shortest distance from A to E via node B (one of A's neighbors) is 3; the shortest distance from node A to E via A's other neighbor D is 2. Thus, in accordance with the

previous paragraph, all A needs to know about E is that via B it is 3 hops away, and via D it is 2 hops away. Similarly, all A needs to know about C is that it is 2 hops away via B, and 2 hops away via D. Similarly for nodes B and D.

The above mentioned information can be stored in tabular form, as shown in Fig. 4b. Such a table, called the Distance Table, has a column for every neighbor of the node in question, and a row for every node in the network. Each node has such a distance table, and it should be clear that each node's distance table will be different from those of the other nodes.



(A)

NODE D

DISTANCE TABLE				ROUTE TABLE
	DA	DC	DE	
A	1	3	3	1,A
B	2	2	3	2,A
C	3	1	2	1,C
D	-	-	-	- -
E	3	2	1	1,E

NODE A

DISTANCE TABLE			ROUTE TABLE
	AB	AD	
A	-	-	- -
B	1	3	1,B
C	2	2	2,B
D	3	1	1,D
E	3	2	2,D

(B)

Fig. 4 Examples of Distance and Route Tables

Both for expository ease and for programming reasons we now introduce a second table, the Route Table. It is structured to take advantage of the fact that the information usually needed is the shortest route to a particular node. The route table has an entry for every node in the network. The entry for node X contains the shortest distance to node X, and the name of the first node on the route along which the shortest distance obtains. Thus the route table in node A (Fig. 4b) has "2,D" in the entry for node E, since the shortest route from A to E is AD, DE. It should be clear that the route table can easily be constructed from the information in the distance table; and once again, the route table in each node is different from those in the other nodes.

When the topology of the network changes, the distance and routing tables in some (or perhaps all) of the nodes have to be updated. This update information is disseminated by means of the NETCHANGE messages. The general idea behind the NETCHANGE messages runs something like this.

When a node discovers that one of its immediate neighbors is no longer a neighbor (either because the link between them went down or because the neighbor crashed), or conversely when it discovers that it has a new neighbor, the node updates its distance and route tables, and then sends the updates to all its neighbors. In the case of a new neighbor, it also sends a copy of the route table to that new neighbor. These updates and the copy of the routing table are the NETCHANGE messages. These neighbors will in turn use the information in these NETCHANGE messages to update their distance and route tables. If this causes any entries of their route tables to change, i.e., if the update changed the shortest distance to some node, then the neighbor in question would send these changes to all its neighbors in the form of NETCHANGE messages. The process continues until all the tables of all nodes have been correctly updated.

Now it is quite possible that while the above process is going on, another link goes down or comes up, or another node crashes or comes up. This will cause another stream of NETCHANGE messages, which will mingle with the first stream. We have proved that this causes no problems and have discussed this in another report.

Before going into this in more detail, we would like to point out a basic fact about the NETCHANGE process. The distance table of each node consists of nothing more than a collection of the information in the route tables of all the neighbors of that node. NETCHANGE messages merely insure that a node is informed of changes in its neighbors' route tables.

The NETCHANGE Protocol Detailed Specification

A NETCHANGE message "SN ON, SD" has the following meaning: The (new) shortest distance from the sender of the message SN to some other node ON is SD.

Notations used in the following explanation

Node B: node carrying out the algorithm
Node C: A neighbor of node B.
Node Y: Some node other than B or C.
N: The number of nodes in the network.

There are three events that can occur to node B:

1. An adjacent link BC comes up, or a neighbor node C comes up.

2. An adjacent link BC goes down, or a neighbor node C goes down.

3. A NETCHANGE message [CY,D] is

received (from node C).

Note that, as in the adjacency matrix scheme, a node going down is treated as if all the links to that node were going down.

The following three algorithms describe what a node does when one of the three possible events occur.

Algorithm #1: An adjacent link (BC) or neighbor (C) comes up.

1. Entry (C,BC) of the distance table is set to 1.
2. Entry (C) of the route table is set to 1,C.
3. The NETCHANGE message [BC,1] is sent to all neighbors of B.
4. A copy of B's route table is sent to C in the form of a series of NETCHANGE messages.

Figure 5 gives an example of nodes executing algorithm 1. Note that the number N (N=5 in this example) is used to indicate the lack of a path, since the longest path without loops in a network of N nodes can be no longer than N-1 links. Thus when a link is down, the corresponding column in the distance table of an adjacent node is filled with N's.

Algorithm #2: An adjacent link (BC) or neighbor (C) goes down.

1. All entries in column BC of B's distance table are set to N. Let X take on the value of every node ID except B, and for each value of X:
2. Examine row X of the distance table to see if step 1 increased the minimum distance to node X. If it did not, do nothing further for this value of X.
3. If it did, call the new value D'. Make the appropriate change to entry X of the route table, and send the NETCHANGE message [BX,D'] to all neighbors.

Algorithm #3: A NETCHANGE message [CY,D] is received.

1. If Y=B, the message is accepted by node B, but then it is simply ignored. If Y=B, go to next step.
2. Entry (Y,BC) of B's distance table is set equal to Min(D+1,N).
3. Row Y of the distance table is examined to see if step 2 changed the shortest distance to node Y. If not, nothing further is done. If the distance is changed, go to step 4.

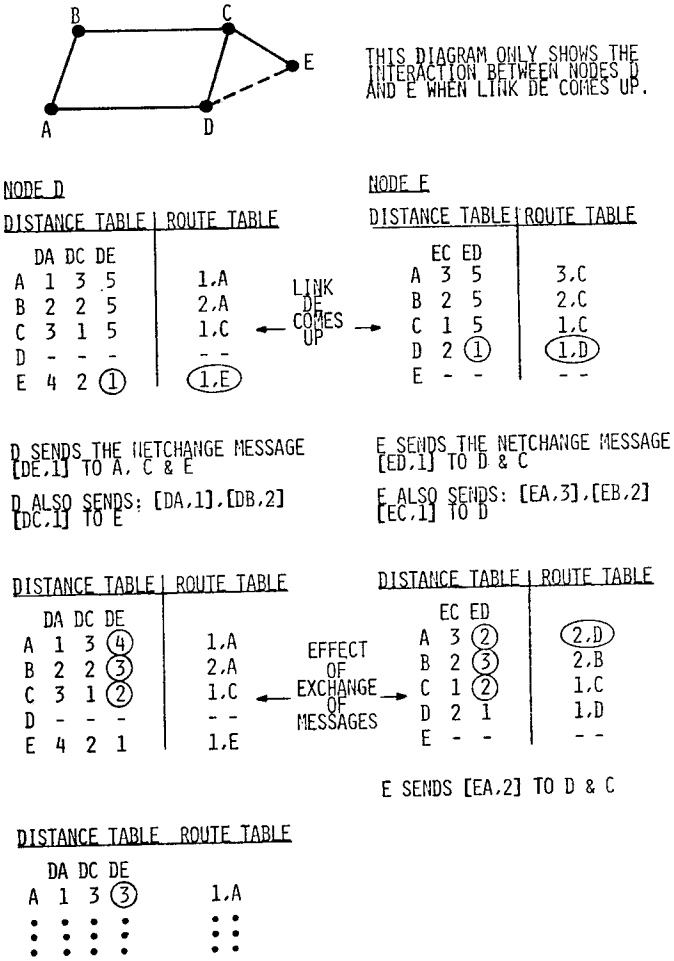


Fig. 5 Example showing part of the NETCHANGE message activity induced by a topology change

4. Let the new shortest distance be D'. The appropriate change is made to entry Y of the route table, and the NETCHANGE message BY,D' is sent to all neighbors.

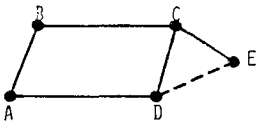
Figure 6 gives an example of nodes executing algorithm #3 (and parts of algorithm #1).

The activity in a network when a link goes down is very much like the activity when a link comes up, except the link's going down can cause one or more nodes to become isolated from the network. When this happens, NETCHANGE message traffic is generated causing the entries in row Y of the distance tables (where Y is the isolated node) to percolate up to the value N.

When a node is initialized, all the distance table entries are set equal to N.

We implemented the NETCHANGE message

protocol, in the MERIT Network in the summer of 1973. It was easy to install and has been absolutely reliable. In



THIS DIAGRAM ONLY SHOWS THE INTERACTION BETWEEN NODES A, B, C, & D.

NODE D'S TABLES CHANGE AS SHOWN IN FIG. 5. IT SENDS THE NETCHANGE MESSAGE [DE,1] TO A & C:

NODE A

DISTANCE TABLE			ROUTE TABLE
	AB	AD	
A	-	-	- -
B	1	3	1,B
C	2	2	2,B
D	3	1	1,D
E	3	3	3,B

TABLES PRIOR TO CHANGE

NODE C

DISTANCE TABLE				ROUTE TABLE
	CB	CD	CE	
A	2	2	4	2,B
B	1	3	3	1,B
C	-	-	-	- -
D	3	1	3	1,D
E	3	3	1	1,E

A & C RECEIVE [DE,1] FROM D.

DISTANCE TABLE			ROUTE TABLE
	AB	AD	
A	-	-	- -
B	1	3	1,B
C	2	2	2,B
D	3	1	1,D
E	3	(2)	(2,D)

DISTANCE TABLE				ROUTE TABLE
	CB	CD	CE	
A	2	2	4	2,B
B	1	3	3	1,B
C	-	-	-	- -
D	3	1	3	1,D
E	3	(2)	1	1,E

NOW NODE A SENDS [AE,2] TO B & D:

NODE B

DISTANCE TABLE			ROUTE TABLE
	BA	BC	
A	1	3	1,A
B	-	-	- -
C	3	1	1,C
D	2	2	2,C
E	4	2	2,C

NODE D

DISTANCE TABLE				ROUTE TABLE
	DA	DC	DE	
A	1	3	3	1,A
B	2	2	3	2,A
C	3	1	2	1,C
D	-	-	-	- -
E	4	2	1	1,E

B & D RECEIVE [AE,2] FROM A

DISTANCE TABLE			ROUTE TABLE
	BA	BC	
A	1	3	1,A
B	-	-	- -
C	3	1	1,C
D	2	2	2,C
E	(3)	2	2,C

DISTANCE TABLE				ROUTE TABLE
	DA	DC	DE	
A	1	3	3	1,A
B	2	2	3	2,A
C	3	1	2	1,C
D	-	-	-	- -
E	(3)	2	1	1,E

Fig. 6 Further NETCHANGE message activity induced by the change of Fig. 5

fact, we have proved that the NETCHANGE protocol is correct.

We only have space here to sketch our correctness proof. The proof borrows a technique from automata theory. First, we show that if the protocol terminates, it terminates correctly, then we show that it always terminates.

The proof of correct termination is based on the structure of the route and distance tables. Entry Y of B's route table is based on row Y of B's distance table, which in turn is based on the Y entries of the route tables of B's neighbors. One can thus trace a path back to some neighbor of Y, which is always correct about its distance from Y.

The proof of certain termination is more complicated. It is based on the observation that a NETCHANGE message with ON=Y can only affect row Y of a distance table, and can only cause the generation of other NETCHANGE messages that also have ON=Y. By confining our attention to NETCHANGE messages on "level" Y (i.e., with ON=Y), we were able to show that a finite number of topology changes could not cause an infinite number of messages to be generated on level Y, and hence could not cause an infinite number of NETCHANGE messages to be generated.

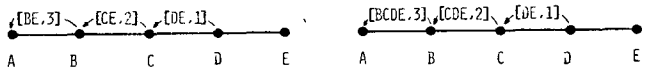
Extensions of the NETCHANGE Protocol

The reader will recall that each node has limited information about the network topology, to wit:

1. the identity of its neighbors.
2. for every node Y, the distances from itself to Y via each of its neighbors.

This information is sufficient for our purposes, but may not be enough in other cases. However, it turns out that our NETCHANGE Protocol can be modified to provide the nodes with more information.

In some applications, the identity of the nodes along the shortest paths is important information. By modifying the NETCHANGE messages so that they contain not only SN and ON, but also the ID's of the nodes in between, the extra information is maintained inside each node (Fig. 7). Thus our NETCHANGE Protocol can easily be turned into a "shortest path" algorithm.



EXAMPLE OF MESSAGE FLOW USING UNMODIFIED PROTOCOL.

EXAMPLE OF MESSAGE FLOW USING A MODIFIED PROTOCOL THAT PROVIDES EACH NODE WITH A COMPLETE DESCRIPTION OF THE SHORTEST PATHS.

Fig. 7

By applying a similar straightforward modification, it is possible to maintain in each node a list of all paths from it to any other node (this implies having the identity of all the nodes along each such path).

We conclude by noting that our NETCHANGE Protocol is only the first step in the development of an effective message routing scheme. Other problems that remain are flow control and congestion control. Flow control is necessary to synchronize communicating processes in different nodes; one of its functions is to prevent a receiving process from being flooded with messages. Congestion control is necessary to route

messages around congested parts of the network, and to prevent congestion whenever possible. We plan to pursue research on these subjects.

Acknowledgments

I would like to thank Wayne Fischer and Eric Aupperle for their helpful insights during the course of this research.

Bibliography

1. Aupperle, E. M., "The MERIT Network Re-examined," Report No. MCN-0273-TP13, MERIT Computer Network, University of Michigan, Ann Arbor, Michigan, February 1973.

2. Baran, Paul "On Distributed Communication Networks," IEEE Transactions on Communication Systems V CS-12, March 1964, pp. 1-9.

3. Beere, Max P. and Sullivan, Neil C., "TYMNET-A Serendipitous Evolution," IEEE Transactions on Communications, V COM-20 #3, June 1972, pp. 511-515.

4. Cocanower, A. B. et al., "The Communications Computer Operating System--The Initial Design," MERIT Computer Network, University of Michigan, October 1970.

5. Dreyfus, Stuart E. "An Appraisal of Some Shortest-Path Algorithms," Operations Research, V17, #3, May-June 1969, pp. 395-412.

6. Heart, F. W. et al., "The Interface Message Processor for the ARPA Computer Network," AFIPS Conference Proc., V36, 1970 SJCC, pp. 551-567.

7. Herzog, B., "Computer Networks," Proc. of the International Computing Symposium, Venice Italy, 12-14 April 1972.

8. Frank, Howard, Kahn, R. E., and Kleinrock, Leonard, "Computer Communication Network Design-Experience with Theory and Practice," AFIPS Conference Proc., V40, 1972 SJCC, pp. 255-270. (Contains an extensive bibliography on computer networks.)

9. Tajibnapis, William D., "A Correctness Proof of a Topology Information Maintenance Scheme for a Distributed Computer Network," submitted for publication to the Communications of the ACM.