References

1. Baskett, F., and Muntz, R.R. Queueing network models with different classes of customers. Proc. Sixth Annual IEEE Int. Conf., San Francisco, Sept. 1972, pp. 205-209.

2. Bernstein, A.J., and Sharp, J.C. A policy-driven scheduler for a time-sharing system. *Comm. ACM 14*, 2 (Feb. 1971), 74-78.

3. Chua, Y.S., and Bernstein, A.J. Analysis of a feedback scheduler. *SIAM J. Comptg. 3*, 3 (Sept. 1974), 159-176.

4. Coffman, E.G. and Kleinrock, L. Computer scheduling methods and their countermeasures. Proc. AFIPS 1968 SJCC, Vol. 32, AFIPS Press, Montvale, N.J., pp. 11-21.

5. Coffman, E.G., Elphick, M.J., and Shoshani, A. System deadlocks. *Computing Surveys 3*, 2 (June 1971), 67-78.

6. Coffman, E.G., and Kleinrock, L. Feedback queueing models for time-shared systems. *J. ACM 15*, 4 (Oct. 1968), 549-576.

7. Coffman, E.G. Analysis of two time-sharing algorithms designed for limited swapping. *J. ACM 15*, 3 (July 1968), 341-353.

8. Denning, P.J. The working set model for program behavior. *Comm. ACM 11*, 5 (May 1968), 323-333.

9. Feller, W. *An Introduction to Probability Theory and Its Applications, Vol. I.* Wiley, New York, Third Ed., Rev. Printing, 1970.

10. Greenberger, M. The priority problem and computer time sharing. *Manage. Sci. 12*, 11 (July 1966), 888-906.

11. Kleinrock, L. A continuum of time-sharing scheduling algorithms. Proc. AFIPS 1970 SJCC, Vol. 36, AFIPS Press, Montvale, N.J., pp. 453-458.

12. Kleinrock, L. Time-shared systems: A theoretical treatment. *J. ACM 14*, 2 (April 1967), 242-261.

13. Kleinrock, L. A delay dependent queue discipline. *Nav. Res. Log. Quart. 11*, 4 (1964), 329-341.

14. Kleinrock, L., Muntz, R.R., and Hsu, J. Tight bounds on the average response time for time-shared computer systems. Information Processing 71, North-Holland Pub. Co., Amsterdam, pp. 124-133.

15. Lynch, H.W., and Page, J.B. The OS/VS2 release 2 system resources manager. *IBM Systems J. 13*, 4 (1974), 274-291.

16. McKinney, J.M. A survey of analytical time-sharing models. *Computing Surveys 1*, 2 (June 1969), 105-116.

17. Ruschitzka, M. System resource management in a time sharing environment. Ph.D. Th., Dept. of EECS, U. of California, Berkeley, Nov. 1973.

18. Schrage, L.E. The queue M/G/1 with feedback to lower priority queues. *Manage. Sci. 13*, 7 (1967), 466-474.

Computer Systems

C. Bell, D. Siewiorek, and S.H. Fuller, Editors

A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network

William D. Tajibnapis
MERIT Computer Network

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 topology information scheme is that it may not involve a central controller. The Topology Information Protocol that was implemented on the MERIT Computer Network is presented and explained; this protocol is quite general and could be implemented on any computer network. It is based on Baran's "Hot Potato Heuristic Routing Doctrine." A correctness proof of this Topology Information Protocol is also presented.

Key Words and Phrases: distributed computer network, correctness proofs, computer networks, distributed control, network topology, routing problem in networks, distributed operating system, store and forward packet switching, store and forward message switching, traffic control

CR Categories: 3.81, 4.32

Author's present address: Soo Line Railroad, Soo Line Building, Minneapolis, MN 55440.

Introduction

In this paper we present a problem that the designers of a distributed computer network must solve. We then describe a solution to that problem and proof of its correctness. We would like to point out that our descriptions of both the problem and its solution are quite general and hardware independent; the solution we propose can be easily implemented in most computer networks. In particular, it has been implemented by us on the MERIT Computer Network [1, 3, 6].

The MERIT Computer Network, jointly funded by the three participating universities, is intended to connect the computers of many of the state's universities and colleges. At present it has three nodes which are located about 70 miles from each other: Michigan State University, The University of Michigan, and Wayne State University.

A typical (ARPANET-like) [4, 5] computer network is a collection of host computers connected by a communication subnet. The communication subnet is a collection of minicomputers (the *nodes* of the network) interconnected by bidirectional electronic *links* (usually telephone lines). Each minicomputer is either connected to one or more host computers or is itself an entry point to the network. The purpose of such a network is to connect the host computers so that a user of any one of these host computers or one of the entry points has access to all the host computers on the network. MERIT is such a network.

The communication minicomputers are essentially invisible to the user; the reason for having all that extra hardware is to reduce the host software costs. Without them it would be necessary to interface every host computer with every other host computer, and to install telecommunications support in every host. With them, it is only necessary to interface each host with the minicomputer, and the (identical) minicomputers with each other.

Finally, both APPANET and MERIT are distributed networks. This means that all nodes are equal and there is no node that is a central controller.

Our solution (to the as yet undescribed problem) is designed for a distributed network with equal nodes, but could also be implemented on a centralized network. It should also be noted that it is a matter of indifference to the discussions which follow whether the nodes are minicomputers or the hosts themselves. Such networks, in which the hosts are directly connected to the communication lines, do exist. Typically, all hosts on such a network are very much alike (e.g. all members of a particular computer series).

On the other hand, our solution will not work for a network in which the links are unidirectional. Thus a ring network in which the messages can only move clockwise does not fall within the scope of this article.

We conclude this introduction by introducing some terms that we will need. A *node* is a (mini)computer. A *link* XY is a bidirectional physical connection between the two nodes X and Y. Node X is a *neighbor* of node Y if (i) both X and Y are operational (up), (ii) the link XY exists *and* is operational. A *route* between two nodes X and Y is a connected sequence of operational links and nodes, starting with some link XP and ending with some link QY. Thus in Figure 1, $ABCE$, $ABCDCE$ are routes, but ABD, AEC are not. A *user* is a process or job in some host; a record is the unit of user-generated information that the network conveys from *host X* to *host Y*. A *message* is a network-generated piece of information that travels from some *node X* to some *node Y*.

The Problem

In a network of any size at all, it quickly becomes quite impractical to have a direct connection between every two nodes. Thus in order for the network to route records between nodes A and B, it may be necessary to route these records via nodes C, D, and E. This implies that:

(1) Node A (the sender node) must know enough about the network topology to determine that in order to get the record to node B it should send it to node C.

(2) Node C (the intermediate node) must recognize that the record is not destined for itself, but for some other node. Node C must also know enough about the network topology to determine that in order to get the record to node B it should send it on to node D, its neighbor.

(3) When the network topology changes,[1] some scheme must exist whereby these changes are made known to all nodes in the network. The occurrence of topology changes should be invisible to the users of the network; for example, a person at node A communicating with the computer at node B should be totally unaffected by changes (which occur while he is "on") in the route between nodes A and B, as long as such a route continues to exist. Clearly, if all routes between A and B are down, the user's connection should be automatically closed and the user should be informed of what happened.

A. The Netchange Message Protocol

The necessary information about the topology of the network is stored in (routing) tables. The nodes in the network inform each other about topology changes by sending each other special messages called NETCHANGE messages. Thus there are routines that send and receive NETCHANGE messages and make the appropriate changes in the routing tables.

The actual process of record relaying (or record switching) is a conceptually trivial one; the sender and

[1] A "topology" change is defined to be any one of the following events, or any combination thereof: (1) A link going down (i.e. ceasing to function), (2) a link coming up, (3) a node crashing or restarting (by this is meant the node minicomputer, not the host).

intermediate nodes merely consult their routing tables to determine which neighboring node to send the record to. The interesting part is the scheme used to make network topology changes known to all nodes in the network (hereafter called the NETCHANGE Protocol).

Thus the problem is to design a correct topology information maintenance protocol. Our solution is a topology information maintenance protocol called the NETCHANGE protocol. The remainder of this article will describe the NETCHANGE Protocol that we have implemented and prove its correctness.

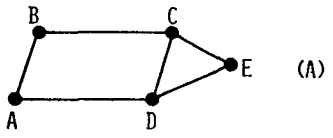
Incidentally, we would like to make clear at the outset that our NETCHANGE Protocol will work for a network of arbitrary size, not just for our three-node network. It is based on Baran's "Hot-Potato Heuristic Routing Doctrine" [2].

We mentioned in the previous section that the "necessary information about the topology of the network is stored in the routing tables." What precisely is this necessary information?

In order to correctly relay (or send) a record, a node need not know the complete, exact topology of the network. All it needs to know is what direction to send the record, i.e. what link it should transmit the record on. Another way of saying the same thing is that 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 protocol the node knows the absolute length of these routes. Length is measured in terms of "hops": thus a route with two intermediate nodes in it is three hops along.

For example, consider node A in the network of Figure 1. The shortest distance from A to E via node B (one of A's neighbors) is 3, since there are three links in that route: AB, BC, CE. The shortest distance from node A to E via A's other neighbor, D, is 2, since there are two links in that route: AD, DE. Thus, in accordance with the previous paragraph, all A needs to know about E is that via B it is three hops away, and via D it is two hops away. Similarly, all A needs to know about C is that it is two hops away via B, and two hops away via D. Similarly for nodes B and D.

Fig. 1.



(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)

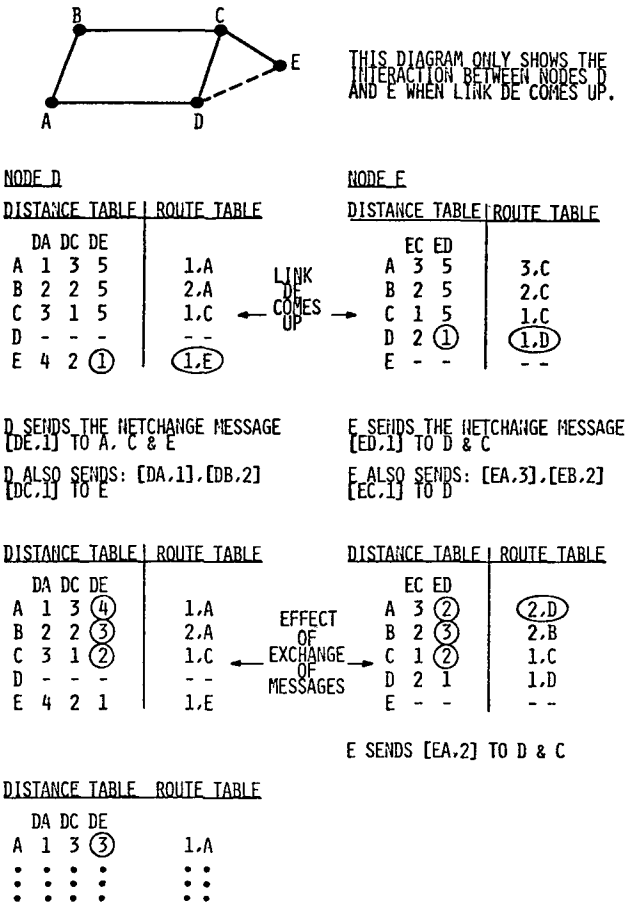
The above-mentioned information can be stored in tabular form, as shown in Figure 1(b). Such a table, called the Distance Table, has a column for every neighbor of the node in question, and a row for every node in the network. Each node has such a distance table, and it should be clear that each node's distance table will be different from those of the other nodes. Figure 1(b) shows the Distance Table for node A; entry (E, AB) contains a 3 since the distance from A to E via B is three hops. Similarly, entry (E, AD) contains a 2 since the distance from node A to node E via D is two hops. Figure 1(b) also shows the Distance Table for node D; note that it has three columns, since node D has three neighbors.

Both for expository ease and for programming reasons we now introduce a second table: the Route Table. It will be needed later on in our proofs, and it is structured to take advantage of the fact that most often the information 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 (see Figure 1(b)) 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 the following.

When a node discovers that one of its 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 (more of which later). 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 the *Route* Table to change, i.e. if the update changed the shortest distance to some node, then the neighbor in question will send these changes (or updates) to all its neighbors in the form of NETCHANGE messages. The process continues until all

Fig. 2.



nodes' tables 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 will prove (later on) that this causes no problems.

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 ensure that a node is informed of changes in its neighbors' Route Tables; NETCHANGE messages are only sent from one neighbor to another. We would also like to point out that no assumptions are made about the relative processing speeds of the nodes, or transmission rates of the links.

The NETCHANGE Protocol: Detailed Specification. A NETCHANGE message "TID OID, SD" has the following meaning: The (new) shortest distance from the sender of the message TID (*T*his node's *ID*) to some other node OID (*O*ther node's *ID*) is *SD* (*S*hortest *D*istance).

The following notation is used in the explanation:

Node B: The node that is carrying out the algorithm.

Node C: A neighbor of node B.

Node Y: Some node other than B or C.

NN: The number of nodes in the network. (Nodes that happen to be down are included in this count.) This is a number that is known to all nodes.

Distance Table: The entries[2] of this table will be referred to as (y, bc). The actual contents of an entry (which is one number, the distance from B to Y via neighbor C) will be referred to as $D_{y,bc}$.

Route Table: Each entry[2] consists of a shortest distance and the ID of the neighbor along which that shortest distance obtains. The entry for node P will be referred to as S_p, N_p.

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) and 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 a node going down is treated as if all the links to that node were going down (in practice, these two events often cannot be distinguished).

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

Algorithm 1 (A link comes up)

When node B discovers that link bc has come up (or that node C has come up) it does the following:

1. Entry (c, bc) of the Distance Table is set to 1.
2. Because this action will always change the shortest distance from B to C from some $d > 1$ to 1, 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. Let the entries of B's Route Table be numbered $p, q, r, \ldots, t$. Then B sends the NETCHANGE messages $[bp, S_p]$, $[bq, S_q]$, $[br, S_r]$, $\ldots$, $[bt, S_t]$ to node C.

Figure 2 gives an example of nodes executing Algorithm 1. Note that the number NN (5 in this example) is used to indicate the lack of a path, since the longest route without loops in a network of NN nodes can be no longer than $NN - 1$ links. Thus when a link is down, the corresponding columns in the Distance Tables of the two adjacent nodes are filled with NNs.

Algorithm 2 (A link goes down)

When node B discovers that link BC has gone down (or that node C has crashed) it does the following:

1. All entries in column bc of B's Distance Table are set to NN. Then for each row p of the Distance Table (and Route Table) the following is done:
2. Let the columns of the Distance Table be $bf, bg, \ldots, bc, \ldots, bk$. The minimum of $D_{p,bf}, D_{p,bg}, \ldots, D_{p,bc}, \ldots, D_{p,bk}$ is computed. (This minimum may have changed because $D_{p,bc}$ was changed to NN.) Let this minimum be $D_{p,bg}$.
3. If $D_{p,bg} = S_p$ (the Route Table entry for row p) set $N_p = G$. Otherwise set S_p equal to $D_{p,bg}$, set N_p equal to G and send the NETCHANGE message $[bp, S_p]$ (where S_p is the new value) to all neighbors.

[2] The entries of the Distance Table are initialized to NN. The entries of the Route Table are initialized to "A,NN" where A is an arbitrary node ID. (See the Appendix about the details of network initialization.)

Algorithm 3 (A NETCHANGE message $[CY, D]$ is received)

If $Y = B$, the message is accepted by node B, but then is simply ignored. If $Y \neq B$, the following steps are taken:

1. $D_{y,bc}$ is set equal to Min $(D + 1, NN)$.
2. Compute the minimum of $D_{y,bf}, \ldots, D_{y,bg}; \ldots, D_{y,bk}$. Let it be $D_{y,bg}$.
3. If $D_{y,bg} = S$ set $N = G$. Otherwise, set S_y equal to $D_{y,bg}$ and set $N_y = G$ and send the NETCHANGE message $[by, S_y]$ to all neighbors.

Figure 3 gives an example of nodes executing Algorithm 3 and parts of Algorithm 1.

The activity in a network when a link goes down is exactly analogous to what happens when a link comes up, as long as the loss of that link doesn't cause a node to become isolated. So we will not present a separate example for that case. We will, however, present an example of the case of a link going down and isolating a node.

For this example, we will consider what happens to the network of Figure 2 (without link DE) when link CE goes down (see Figure 4). The going down of this link will only affect distances between E and some other node, and not distances between two nodes X and Y, where neither X nor Y are E. Thus in order to simplify matters, in Figure 4 we will only show row E of the various Distance Tables and Route Tables.

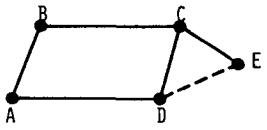
Clearly, node E immediately finds out that it is isolated, and since it no longer can communicate with any other nodes, its activities are of no further concern to us.

The first row of Figure 4 shows the tables of nodes A, B, C and D prior to CE's going down. When link CE goes down, node C takes cognizance of the event by putting 5's in column CE of its Distance Table. This changes C's shortest distance to E from 1 to 3, so the Route Table entry for E changes from 1, E to 3, B (see the second row of Figure 4). Then C sends the NETCHANGE message "CE, 3" to its neighbors B and D.[3, 4]

Thus even though C knows that link CE went down, it does not yet know that node E is now unreachable. This is because C does not know the complete topology of the network, and for all it knows B might be connected to E. What happens now is that nodes A, B, C, and D send each other NETCHANGE messages which cause their Route Table entries for node E to climb up to $5, X$. The reader will recall that since there are five nodes in the network, a distance of 5 is interpreted to mean that the node in question is inaccessible.

This example concludes our description of the NETCHANGE Protocol. In the next part of this article, we will study the correctness of this protocol.

Fig. 3.



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

NODE D'S TABLES CHANGE AS SHOWN IN FIG. 2. 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

B. Correctness Proof of the NETCHANGE Protocol

1. Decomposition of the network. We now introduce a somewhat different model of the network that will be used in some of our proofs. This new, decomposed model, is more general than the model described in the previous section; however, by imposing some restrictions that do not affect our proofs, we can make the decomposed model and the old model exactly equivalent.

This new model was inspired by an interesting property of Algorithm 3. When this algorithm receives a NETCHANGE message with OID = y (OID stands

[3] Normally a node adds 1 to a distance in a NETCHANGE message it receives before putting it in its Distance Table. But if the distance in the message is 5, the 5 is put directly in the Distance Table. This is because 5 stands for infinity or inaccessibility.

[4] It is essential for the correct working of the NETCHANGE Protocol that no NETCHANGE messages are lost due to transmission errors or noise on the lines (links). This can be ensured by having the NETCHANGE messages acknowledged. (The acknowledgment protocol is used between neighbor nodes to correct transmission errors and prevent lost messages. Typically, the procedure is for the sender to resend the message at regular intervals until it receives an acknowledgment from the receiver. For an extensive discussion of this protocol, see Chapter 5 of [7]). On the other hand, it is all right for a NETCHANGE message to get lost because the link on which it is traveling goes down.

for "Other Node's ID"), it only changes row y of the Distance Table, and possibly entry y of the Route Table. Similarly, if it sends out NETCHANGE messages, these messages will all have OID = y.

Since we are not concerned with the order in which NETCHANGE messages are processed or sent by a node, it seemed to us that Algorithm 3 could be replaced by NN parallel algorithms, each of which is responsible for one row of the Distance Table and one entry of the Route Table.

Furthermore, Algorithms 1 and 2 can be similarly altered. Since the three algorithms together constitute one process, the three decomposed algorithms together constitute NN parallel processes (in one node[5]). A description of these decomposed algorithms follows.[6]

Algorithm 1 (in node b for level $y(y \neq b)$: a link comes up)

When link bc or node c comes up, do the following (c is some neighbor of b).

1. If $C = Y$: Set D_y, b_y equal to 1
   Set $S_y = 1$, $N_y = y$
   Send the NETCHANGE message $[b_y, 1]$ to all neighbors
2. *If* $C \neq Y$, send the NETCHANGE message $[b_y, S_y]$ to node C only.

Algorithm 2 (in node b for level $y(y \neq b)$: A link goes down)

When link bc goes down, or node c crashes, do the following (c is some neighbor of b):

1. Set $D_{y,bc}$ equal to NN.
2. Let the columns of the Distance Table be $bf, bg, \ldots, bc, \ldots, bk$. The minimum of $D_{y,bf}, D_{y,bg}, \ldots, D_{y,bc}, D_{y,bk}$ is computed. Let this minimum be $D_{y,bg}$. (This minimum is computed because it may have changed as a result of step 1.)
3. If $D_{y,bg} = S_y$ set N_y equal to g. Otherwise, set S_y equal to $D_{y,bg}$ and set N_y equal to g; and send the NETCHANGE message $[by, S_{y,bg}]$ to all neighbors.

Algorithm 3 (in node b for level y $(y \neq b)$. A NETCHANGE message is received)

At level y only NETCHANGE messages of the form cy,D are received.

1. $D_{y,bc}$ is set equal to Min $(D + 1, NN)$.
2. Compute the minimum of $D_{y,bf}, D_{y,bc}, \ldots, D_{y,bk}$. Let it be $D_{y,bg}$.
3. If $D_{y,bg} = S_y$ set N_y equal to g. Otherwise, set S_y equal to $D_{y,bg}$ and set $N_y = g$ and send the NETCHANGE message $[by, S_y]$ to all neighbors (where S_y is the new value).

Since the Route and Distance Tables inside node b do not have an entry or row for node b, the process inside node b for level b is particularly simple:

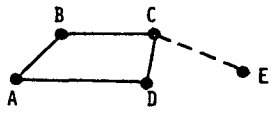
Algorithm 1 in node b for level b: null.
Algorithm 2 in node b for level b: null.
Algorithm 3 in node b for level b: receives NETCHANGE messages of the form $[cb,D]$, then does nothing.

The reader should now note that a level y process

[5] In the interest of clarity, from this point on lower-case letters will be used for node names.

[6] Entries of the Distance and Route Table are still initialized to NN.

Fig. 4.



NODE A			NODE B			NODE C			NODE D		
DISTANCE TABLE		ROUTE TABLE	DISTANCE TABLE		ROUTE TABLE	DISTANCE TABLE		ROUTE TABLE	DISTANCE TABLE		ROUTE TABLE
AB	AD		BA	BC		CB	CD CE		DA	DC DE	
E 3	3	3,B	4	2	2,C	3	3 1	1,E	4	2 5	2,C
			LINK CE GOES DOWN								
E 3	3	3,B	4	2	2,C	3	3 (5)	(3,B)	4	2 5	2,C
			C SENDS CE,3 TO B & D								
E 3	3	3,B	4	(4)	(4,A)	3	3 5	3,B	4	(4) 5	(4,A)
	B	SENDS	BE,4 TO A & C ; D SENDS DE,4 TO A & C								
E(5)	(5)	(5,X)	4	4	4,A	(5)	(5) 5	(5,X)	4	4 5	4,A
	A	SENDS	AE,5 TO B & D ; C SENDS CE,5 TO B & D								
E 5	5	5,X	(5)	(5)	(5,X)	5	5 5	5,X	(5)	(5) 5	(5,X)
	B	SENDS	BE,5 TO A & C ; D SENDS DE,5 TO A & C								
E 5	5	5,X	5	5	5,X	5	5 5	5,X	5	5 5	5,X

(consisting of the Algorithms 1, 2 and 3 for level y) not only accesses exclusively row y of the Distance Table and exclusively entry y of the Route Table, it also exclusively *receives and sends* NETCHANGE messages of the form $[cy, D]$. This means that a level y process can only communicate, directly or indirectly, with other level y processes. Put another way, a level y process cannot influence the behavior of a level x process.

Thus we can think of the network as consisting of NN levels (Figure 5). At level b are found the processes for level b; the only NETCHANGE messages that travel on level b are those of the form $[yb, D]$; similarly on the other levels. Thus in Figure 5, node d at level b represents the level b process in node d. Naturally, the link ad at level a, and the link ad at level b, and the links ad at levels c, d, and e all map into the one physical link ad.

When a topology change occurs, it will cause NETCHANGE message activity on one, all or some of the levels. In the example of Figures 2 and 3, link de coming up caused activity on all levels (because of the exchange of Route Tables between nodes d and e). In the example of Figure 4, link ce going down only caused activity on level e (all the processes in nodes c and e were activated, but only those on level e sent any NETCHANGE messages)—that is why the only changes that took place in the various Route and Distance Tables were in row e.

The reason for decomposing the network like this is that it enables us to focus our attention upon a single level. A proof about the activity on a particular level will apply to all levels, and thus will apply to the activity of the whole network. In the remainder of this article we shall do just that: analyze the activity on a particular level.

At this point, we should qualify one of our previous

statements; the statement that a level x process cannot influence a level y process. There is *one* way in which such influence can occur; since all NETCHANGE messages, no matter what level they are on, travel along the same physical links, a level x message could delay a level y message. As might be expected, neither the NETCHANGE protocol nor any of our proofs about it are timing dependent. Hence the nature and extent of this indirect interaction can be ignored.

We can make an even stronger statement. Various not-so-minor details about the 'software' inside each node are left unspecified in the decomposed description of the NETCHANGE protocol; for example, nothing has been said about the number of processors in each node and how they (or it) are assigned to the *NN* processes. However, the level of detail of our description is sufficient for the proof of correctness that we are going to present.

This means that our correctness proof will be valid for any implementation of the "decomposed" NETCHANGE protocol; where by implementation is meant the specification of the details described in the previous paragraph. It should now be evident that the NETCHANGE protocol, as we described it in Section A, is in fact such an implementation. Thus, in particular, our correctness proof will be valid for the NETCHANGE protocol as described in Section A.

2. The Proofs. We have already introduced the convention that we will be using lower-case letters for node IDs, instead of upper-case letters as in Section A. We now introduce a second convention: All NETCHANGE messages will be referred to just as messages.

We remind the reader that a NETCHANGE message has three fields:

TID: This Node's ID (the sender node's ID)
OID: Other Node's ID
SD: Shortest Distance

The meaning of the message is that the shortest distance from node TID to node OID is now SD.

We also remind the reader that a link is identified by its adjacent nodes; e.g. *ad* is the link between nodes a and d.

Definition. A message traveling along a link *ad* from a to d will be referred to as ad_i.

Definition. Functional notation will be used to represent part of a message. Thus the SD part of ad_i is referred to as SD(ad_i).

Assumption. A node always knows who its neighbors are. This implies that whenever "1" appears in either the Distance or Route Table, it is always correct (see Algorithm 1). (This assumption has been implicit in our previous discussions, and is explicitly stated here for clarity.)

It should be recalled that two nodes x and y are neighbors if: (i) both node x and y are up, (ii) link xy is operational.

In practice, whenever a node comes up it attempts to initiate a handshaking protocol on all its adjacent links. Similarly whenever a link comes up, the two nodes adjacent to it initiate this handshaking protocol. Part of the information exchanged during this protocol are the node IDs. Two nodes are neighbors, then, if this handshaking protocol is successfully completed.

Our proof of the correctness of the NETCHANGE protocol borrows a technique from automata theory; we will prove that if the algorithm terminates, it always terminates correctly; we then prove that it always terminates. Since the former is easier to prove (in fact it does not explicitly make use of the decomposed network model) we present it first.

THEOREM 1. *If and when all message activity has ceased (no messages on queues, or in the process of being transmitted), all entries in all the Distance Tables will be correct.*

Basically, Theorem 1 makes use of the fact that a Distance Table entry $D_{y,bc}$ in node b is based on the entry S_y in c's Route Table, which in turn is equal to some $D_{y,cd}$ in c's Distance Table. Thus if $D_{y,bc}$ is incorrect, this leads to a sequence of incorrect entries in a sequence of nodes, eventually leading to a neighbor of node y. But a neighbor of y cannot have incorrect entries pertaining to y. Thus there can be no incorrect entries in any Distance Table. The complete, formal proof follows.

PROOF.

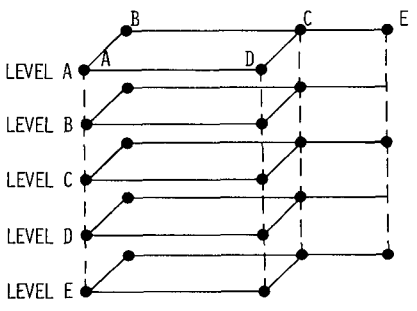
Case 1. The entry $D_{y,bc}$ is equal to 1.
By the above assumption this entry must be correct.

Case 2. $1 < D_{y,bc} < NN$.
We will prove this by contradiction. Assume that the value of $D_{y,bc}$ is too small (the case where it is too large is quite similar).

Since $D_{y,bc}$ is initialized to *NN*, and is set equal to *NN* whenever link *bc* goes down (and/or node c goes down), both link *bc* and node c must be up (see Algorithm 2).

Since $D_{y,bc} \neq 1$ and $D_{y,bc} \neq NN$, it must have received its current value as the result of a message that traveled along link *bc* (see Algorithm 3). Furthermore, the last message that traveled along link *bc* of the form $[cy, D]$ must have had SD = $D_{y,bc} - 1$ (see Algorithm 3).

Fig. 5.



Since there are no further messages in the network (by assumption), this means that entry y in c's Route Table has $S_y = D_{y,bc} - 1$. (See Algorithm 3).

We now shift our point of reference to node c. In its Route Table we find entry $y{:}S_y, N_y$. Let N_y be d. We know that $S_y = D_{y,bc} - 1$, and since $D_{y,bc}$ is too small, this means that S_y is to small. But $S_y = D_{y,cd}$, by Algorithm 3. So now we have an entry $D_{y,cd}$ in c's Distance Table that is too small.

But by repeating the above argument it can be seen that this means that there is some entry $D_{y,de} = D_{y,cd} - 1$ in d's Distance Table which is too small. And by repeated application of the argument we finally reach some node k with an entry $D_{y,km} = 1$ in its Distance Table, and this entry is too small. However, this is impossible, since a node always knows who its neighbors are.

Case 3. $D_{y,bc} = NN$.

We will also prove this by contradiction. Assume that $D_{y,bc}$ is too large (it can't be too small, since NN is the maximum value it can attain). This means that link bc and node c are up; since if either one is down, NN is the correct value for $D_{y,bc}$ (see Algorithm 2). Also, since $D_{y,bc}$ is too large, there is a route $y, \ldots, b, c$ of length less than or equal to $NN - 1$.

By using the argument of Case 2, it can be shown that the S part of the yth entry in c's Route Table must be either NN or $NN - 1$. However, in either case it is incorrect. Therefore it cannot be $NN - 1$, by the proof of Case 2.

Therefore, it must be NN. Therefore, all the entries of row y of c's Distance Table must have the value NN (by Algorithm 3). But then, by repeating the above argument, all neighbors of c have $S_y = NN$ in their Route Tables. But this implies that all nodes with a path to b, except for y, have $S_y = NN$ in their Route Tables. However, this is impossible, since the neighbor(s) of y will have $S_y = 1$ in their Route Tables. □

COROLLARY 1.1. *If and when all message activity has ceased, all entries in all the Route Tables will be correct.*

This corollary follows directly from Theorem 1 and Algorithms 1, 2, and 3.

We now go on to prove that (NETCHANGE) message activity will cease after an arbitrary (but finite) series of topology changes. All of the theorems that follow pertain to message activity on a particular level (level y).

Often a node b will send out the same NETCHANGE message to all its neighbors. The notation bx_i will be used to refer to this group of messages. ("x" will be used as a generic node number.)

Messages are numbered on a per level basis. Thus, the discussion is about level y, message ad_{i+1} is the first message to travel on link ad_i (from a to d), with OID = y after message ad_i (which also had OID = y). There may have been arbitrarily many messages between ad_i and ad_{i+1} with OID $\neq y$.

THEOREM 2. *If a node b receives a message cb_i with $SD(cb_i) = d$, then if a message bx_i is sent out by b as a result of cb_i either $SD(bx_i) = d + 1$ and/or $SD(bx_i) > SD(bx_{i-1})$.*

PROOF. When a node b receives a message cb_i with $SD(cb_i) = d$, it changes the entry $D_{y,cb}$ of its Distance Table to $d + 1$.

1. If this doesn't change $\text{Min}(D_{y,ba}, D_{y,bc}, \ldots, D_{y,bp})$ no new message is sent out.
2. If it decreases this minimum, then $d + 1$ must be the new minimum. Then entry S_y of the Route Table will be changed to $d + 1$, and a new message bx_i will be sent out with $SD(bx_i) = d + 1$.
3. If it increases the minimum, then S_y will be made larger, and a message bx_i with SD equal to the new larger S_y will be sent out. But $SD(bx_{i-1})$ is equal to the old value of S_y. Therefore $SD(bx_i) > SD(bx_{i-1})$. Note that $SD(bx_i)$ could be equal to $d + 1$, hence the and/or in the statement of the theorem. □

COROLLARY 2.1. *If a message bx_i is sent out by node b as result of receiving a message cb_i, then: If $SD(bx_i) < SD(b_{i-1})$ then $SD(bx_i) = Sb(cb_i) + 1$.*

PROOF. Follows immediately from Theorem 2.

THEOREM 3. *Consider a network in which an arbitrary series of topology changes occur between time 0 and t; no changes occur after time t. The network will generate only a finite number of messages with SD = 1.*

PROOF. We will prove the theorem for a particular level y. It follows immediately that the theorem holds for all levels.

By the assumption that a node always knows who its neighbors are, only neighbors of node y can send out messages with SD = 1 (remember that all messages on level y are of the form [xy, SD]). A node only sends out a message when its Route Table entry for y changes. The only way in which this entry can change to 1 is if the node in question becomes a neighbor of y. This can only happen if there is a topology change. Thus if there is only a finite series of topology changes, only a finite number of messages with SD = 1 can be sent out. □

THEOREM 4. *Consider a network in which an arbitrary but finite series of topology changes occur between 0 and time t; no changes occur after time t. Then, a finite time after t, all message activity will cease and all the entries in all the Distance and Route Tables in all the nodes will be correct.*

PROOF. We will simply prove that the message activity will cease in a finite amount of time; the remainder of the theorem follows from Theorem 1 and its corollary. The proof is done by contradiction and induction.

Assume that the message activity never ceases. Then there must be at least one level, call it y, in which an infinite number of messages is generated. Then there must be at least one node which generates infinitely many messages; call this node b.

Since SD can only take on the values 1 through NN, there exists a value v; $1 \leq v \leq NN$ such that there exist infinitely many i such that $SD(bx_i) = v$.

It follows immediately from Theorem 3 that v cannot be 1. Can v be 2? Now a node only sends a message as a result of a topology change or as a result of receiving a message. Since by assumption the number of topology changes is finite, node b must be receiving infinitely many messages from at least one neighbor, call it c.

Then by Theorem 2, one or both of the following must be true for infinitely many i: (a) $SD(bx_i) = SD(cb_i) + 1$, (b) $SD(bx_i) > SD(bx_{i-1})$. But we are assuming $SD(bx_i) = v = 2$. Therefore b must either receive or send infinitely many messages with SD = 1, which is impossible. Therefore v cannot be 2.

Similarly it can be shown that v cannot be 3, 4, ... or NN. Therefore the assumption is false. Therefore the theorem is true. □

Conclusion

We have described and proven the correctness of a protocol for maintaining up to date topology information in the nodes of a network. Several points remain to be clarified.

Although we have proven that NETCHANGE message activity takes only a finite time, we have not shown any upper limits for the number of messages that are generated. We have been able to derive such an upper limit; it is an exponential function of NN, a polynomial function of K (the maximum number of neighbors that a node can have), and a linear function of T (the number of topology changes). Space precludes us from presenting the formulas for this upper limit and its derivation here.[7] Basically the derivation rests on the fact that, on any particular level, a message with $SD = Y$ can cause the generation of at most K messages with $SD = Y + 1$ and at most K messages with $SD > Y + 1$.

We have successfully implemented the NETCHANGE protocol on the MERIT Computer Network.

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.

[7] The derivation of these formulas may be found in Chapter 8 of [7].

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

Appendix: Some Comments on the Initialization of the NETCHANGE Protocol

As mentioned in the first note in Section B, when a node comes up, the entries of its Distance Table are initialized to NN (NN = the Number of Nodes in the Network), and the entries of its Route Table are initialized to "a, NN," where a is an arbitrary node ID.

We have shown in Example 1 in Section A how such a freshly initialized node updates its tables when it joins the network.

Naturally, the larger question arises of how the tables are updated when the network comes up for the "first" time. As it turns out, everything happens quite automatically as the links and nodes come up, and they can come up in any order. (It should be noted that a link is not considered "up" unless both the nodes adjacent to it are also up; this is because the only reliable way of checking the integrity of a link is to carry on a conversation with the node at the other end of that link.)

When a node is initialized, its tables indicate that it is connected to no one in the network. As the links adjacent to that node come up (or as it discovers that these links are operational), the node will update its tables on the basis of information obtained from its new-found neighbors. These neighbor nodes in turn update their tables on the basis of information obtained from their new neighbors, and so eventually all the tables in all the nodes are correctly updated.

Received September 1974; revised May 1976

References

1. Aupperle, E.M. The MERIT network re-examined. Digest of Papers, COMPCON 73, Feb. 1973, pp. 25-29.

2. Baran, P. On distributed communication networks. *IEEE Trans. Comm. Syst. CS-12* (March 1964), 109.

3. Cocanower, A.B., Fischer, W., Gerstenberger, W.S., and Read, B.S. The communications computer operating system – the initial design. No. PB203 552, Nat. Tech. Inform. Service, Springfield, Va., Oct. 1970, p. 94.

4. Frank, H., Kahn, R.E., and Kleinrock, L. Computer communication network design – experience with theory and practice. Proc. AFIPS 1972 SJCC, Vol. 40, AFIPS Press, Montvale, N.J., pp. 255-270 (Contains an extensive bibliography on computer networks).

5. Heart, F.W., Kahn, R.E., Ornstein, S.M., Crowther, W.R., and Walden, D.C. The interface message processor for the ARPA computer networks. Proc. AFIPS 1970 SJCC, Vol. 36, AFIPS Press, Montvale, N.J., pp. 551-567.

6. Herzog, B. Computer networks. Proc. Int. Comptg. Symp., Venice, Italy, April 1972, pp. 12-14.

7. Tajibnapis, W.D. Message-switching protocols in distributed computer networks. Ph.D. Diss., Pub. MCN-0676-TR-22, MERIT Computer Network, Ann Arbor, Mich., 1976.