AD A061 666

DDC FILE COPY

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): A FAILSAFE DISTRIBUTED ROUTING PROTOCOL
5. TYPE OF REPORT & PERIOD COVERED: Technical / rept.
6. PERFORMING ORG. REPORT NUMBER: LIDS-P-852
7. AUTHOR(s): Philip M. Merlin, Adrian Segall
8. CONTRACT OR GRANT NUMBER(s): ARPA Order No. 3045/5-7-57; N00014-75-C-1183; ARPA Ord. 3045
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Massachusetts Institute of Technology, Laboratory for Information & 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: Sep 1978
13. NUMBER OF PAGES: 76
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

78p.

LEVEL

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)

DDC NOV 30 1978

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Routing Network Communication Distributed Computation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An algorithm for constructing and adaptively maintaining routing tables in communication networks is presented. The algorithm can be employed in store-and-forward as well as line switching networks, uses distributed computation, provides routing tables that are loop-free for each destination at all times, adapts to changes in network flows and is completely failsafe. The latter means that after arbitrary failures and additions, the network recovers in finite time in the sense of providing routing paths between all physically connected nodes. Complete rigorous proofs of all these properties are provided.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

410950

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

EE PUB. No. 313
Sep. 1977
Revised May 1978

September 1978

LIDS-P-852

A FAILSAFE DISTRIBUTED ROUTING PROTOCOL

Philip M. Merlin and Adrian Segall
Department of Electrical Engineering
Technion - Israel Institute of Technology
Haifa, Israel.

Abstract

An algorithm for constructing and adaptively maintaining routing tables in communication networks is presented. The algorithm can be employed in store-and-forward as well as line switching networks, uses distributed computation, provides routing tables that are loop-free for each destination at all times, adapts to changes in network flows and is completely failsafe. The latter means that after arbitrary failures and additions, the network recovers in finite time in the sense of providing routing paths between all physically connected nodes. Complete rigorous proofs of all these properties are provided.

The work of A. Segall was supported by the Advanced Research Project Agency of the US Department of Defense (monitored by ONR) under contract No. N00014-75-C-1183.

1. INTRODUCTION

Reliability and the ability to recover from topological changes are properties of utmost importance for smooth operation of data-communication networks. In today's data networks it happens occasionally, more or less often depending on the quality of the individual devices, that nodes and communication links fail and recover; also new nodes or links become operational and have to be added to an already operating network. The reliability of a computer-communication network, in the eyes of its users, depends on its ability to cope with these changes, meaning that no breakdown of the entire network or of large portions of it will be triggered by such changes and that in finite - and hopefully short - time after their occurrence, the remaining network will be able to operate normally. Unfortunately, recovery of the network under arbitrary number, timing, and location of topological changes is hard to insure and little successful analytical work has been done in this direction so far.

The above reliability and recovery problems are difficult whether one uses centralized or distributed rout , control. With centralized routing, one has the problem of central node failure plus the chicken and egg problem of needing routes to obtain the network information required to establish routes. Our primary concern here is with distributed routing that recovers from topological changes; here one has the problems of asynchronous computation of distributed status information and of designing algorithms which adapt to arbitrary changes in network topology in the absence of global knowledge of topology.

The paper presents a distributed protocol that maintains a route from any source to any destination in a network. The protocol is distributed in the sense that no central tables are required and there is no global knowledge of the routes, i.e. each node knows only who is the next node (called the "preferred neighbor") on the route to a given destination. Each node is responsible for updating its own tables (e.g. choosing a new preferred neighbor) and these updates are coordinated by the protocol via control messages sent between adjacent nodes. For a given destination, the set of routes maintained by the protocol are loop-free at all times, and whenever no failures occur, they form a spanning tree rooted at the destination (i.e. a tree that covers all nodes).

To each link in the network, a strictly positive "distance" (or "weight") is assigned which represents the cost of using the link. According to utilization and possibly other factors, this distance may vary with time following long-term trends. The length of any path is the sum of the distances on the links of this path. Destinations may asynchronously trigger the protocol and start update cycles to change routes according to new distances. Such a cycle first propagates uptree while modifying the distance estimates from nodes to the destination and then propagates downtree while updating the preferred neighbors. Each cycle tends to find routes with short paths from each node to the destination, and assuming time-invariance of link weights, the strict minimum (i.e. shortest paths) will be reached within a finite number of update iterations.

The proposed protocol also provides for recovery of routes after failures and for additions of links or nodes to the network. When a link fails, appropriate information is propagated backwards in the network and, in addition, a "request" message is generated and forwarded towards the destination. New links are brought up via a similar protocol. The request message triggers an update cycle and it is guaranteed that within finite time, all nodes physically connected to each destination will have a loop-free route to it. This holds also for multiple topological changes, and even if such changes occur while the protocol is active and the update is in progress. The recoverability of the protocol is achieved without employing any time-out in its operation, a feature which greatly enhances its amenability to analysis and facilitates structured implementation.

The protocol is mainly intended for quasi-static routing in communication networks and the routes provided by the protocol can be used in a variety of ways for actual routing of information. Although specification of information routing algorithms is outside the scope of the present paper, we indicate here a few applications. In a (physical or virtual) line-switched network, it is often impractical to reroute already established conversations, except in case of disruption caused by failure or priority preemption. In this case, the routes provided by the present protocol may be used for assigning paths to new or disrupted calls. For example, in a virtual line-switched network the link weights may represent link delays, and then the path provided by our protocol in steady state will give the minimum delay route for the new call. If the weights represent incremental delay, then the path will minimize network average delay (see [1, eq. (25)]). Other criteria like probability of blocking, can also be taken into consideration in the link weight. Observe that if the link weights change drastically, the above strategy may allow new conversations to follow paths so different from the old ones that together they form a loop, but this is still the best one can do under the constraint that established conversations cannot be rerouted.

Similar strategies can be used in networks using message switching, where the preferred neighbor indicates the first hop of the present best estimated route towards the SINK and the node may for example increase the fraction of messages routed over this path while reducing the fraction sent over other routes. More sophisticated failsafe routing and update procedures, where exact amount of increase and reduction of traffic fractions are indicated so that optimality and routing loop-freedom are achieved, have been obtained using ideas similar to the protocol of this paper and are presented in a subsequent report [2].

Finally, we may mention that the present protocol can replace the simple-minded saturation routing that is presently used in several networks to locate mobile subscribers and to select routing paths [3]. The protocol of this paper has all the advantages indicated in [3, Sec. II] for saturation routing, but requires no time-out and provides a route selected not only on the basis of the instantaneous congestion but on averaged quantities.

This work was inspired by [4] and [5], where minimum delay routing algorithms using distributed computation were developed. These algorithms also maintain a per destination loop-free routing at each step. One of the main contributions of the protocol given in the present paper is to introduce features insuring recovery of the routes from arbitrary topological changes of the network. As a result, the protocol of the present paper is, to our knowledge, the first one that is distributed and for which all the following properties are rigorously proved:

(a) Loop-freedom for routes to each destination at all times.

(b) Independently of the sequence, location and quantity of topological changes, the routes recover in finite time.

(c) Under stationary conditions, the routes converge to paths with minimal weighted length.

Several routing algorithms possessing some of the propert[illegible]ted above have been previously indicated in the literature. In [6], a routing algorithm similar to the one used in the ARPA network [7] but with unity link weights, is presented. It is shown there, that at the time the algorithm terminates, the resulting routing procedure is loop-free and provides the shortest paths to each destination. As with the ARPA routing, however, the algorithm allows temporary loops to be formed during the evolution of the algorithm. The algorithm proposed in [8] ensures loop-free routing for individual messages. This property is achieved by requesting each node to send a probing message to the destination before each individual rerouting;

the node is allowed to indeed perform the rerouting only after having received an acknowledgement from the destination. The extra load on the network by sending probing messages from each node to each destination for each rerouting is clearly extremely large. Also loop freedom for individual messages is a weaker property than loop freedom for each destination. For example, in a three-node network, sending traffic from node 3 to node 1 via node 2 and sending traffic from node 2 to node 1 via node 3 would be loopfree for individual messages, but not loopfree for each destination. See [9] for a more complete discussion of loop freedom.

In addition to the introduction of this particular protocol and the proofs of its main properties, the paper provides contributions in the direction of modeling, analysis and validation of distributed algorithms. The operations required by the algorithm at each node are summarized as a finite-state machine, with transitions between states triggered by the arrival of special control messages from the neighbors, and the execution of a transition may result in the transmission of such messages. Methods for modeling and validation of various communication protocols were proposed in [10] - [13]. These methods are designed however to handle protocols involving either only two communicating entities or nodes connected by a fixed topology. The model we use to describe our algorithm is a combination of these known models, but is extended to allow us to study a fairly complex distributed protocol. The analysis and validation of the algorithm is performed by using a special type of induction that allows us to prove global properties while essentially looking at local events.

Before proceeding, we may mention two other distributed protocols that were recently developed. In [14], an algorithm for network resynchronization is presented and its recovery properties are proved under arbitrary topological changes. A similar goal is obtained by R.G. Gallager in an unpublished work [15], while also determining the paths with minimum number of links between each pair of nodes in the network. Although there is a great similarity between the ways in which the updating information propagates and the distributed computation is performed by the algorithms of [14], [15] and of the present paper, the exact relationship between these protocols is a subject for future research.

2. THE PROTOCOL

To facilitate understanding, we describe the protocol in several steps. We first present the "basic protocol", i.e. assuming that no topological changes occur. Then we describe the additions to the protocol in case of link outage and finally the additions for links becoming operational. A node outage can be represented as the outage of all of its links and similarly, a node becoming operational can be represented as links becoming operational. Therefore, we do not pay special attention to topological changes caused by nodes.

The following comments apply to the rest of the paper:

1. Since we are not concerned with data transfer, we use the term "message" to mean the special control messages employed by the protocol.

2. We assume that messages sent by a node to a neighbor are received in the same order that they are sent, i.e. FIFO is preserved in the links (and local protocols).

3. The protocol proceeds independently for each destination. Consequently, for the rest of the paper we fix the destination and present and analyse the protocol for that given destination, which is denoted by SINK.

2.1 The Basic Protocol

As already mentioned, each node i in the network has at any time a preferred neighbor. Thus, we assume that each node has a variable p_i which points to that neighbor. For the basic protocol, we assume that after initialization, the directed graph defined by the nodes and arcs (i,P_i) form a tree directed towards (and therefore rooted at) the SINK, as exemplified by the network of Fig. 1 where directed arcs denote the preferred neighbors $\{p_i\}$. Subsequent sections describing the protocol which handles topological changes will show that this assumption is justified by the initialization procedure. Each node i has also a positive variable d_i maintained by the protocol, denoting an estimated distance from i to the SINK (d_{SINK} is by definition equal to 0). During an update, the protocol reevaluates the distances $\{d_i\}$ and accordingly the nodes choose preferred neighbors $\{p_i\}$ in such a way that the directed graph given by the arcs (i,p_i) remains at all times a tree rooted at the SINK.

As already mentioned in Section 1, to each link (i,ℓ) a strictly positive "distance", denoted by $d_{i\ell}$, is assigned. We assume all links to be full duplex and allow a link to have a different distance in each direction. The distance $d_{i\ell}$ is allowed to vary with time and needs to be known (measured or estimated) only by node i. The protocol tends to minimize the distance d_i from each node i to the SINK, where this distance is given by the sum of the weights $d_{\ell m}$ on the directed path from a node to the SINK.

As described below, the SINK may asynchronously start update cycles to change routes according to new distances. Such a cycle first modifies distance estimates $\{d_i\}$ uptree and then modifies preferred neighbors $\{p_i\}$ downtree. An update cycle is started by the SINK by sending a message $MSG(d_{SINK})$ to each of its neighbors (notice that $MSG(d_{SINK}) = MSG(0)$ by definition). When a node, say i, receives a message from its p_i, it reevaluates its estimated distance d_i and transmits $MSG(d_i)$ to each of its neighbours except p_i. Notice that the spanning tree structure mentioned before (Fig. 1) guarantees that after the SINK has started the updating cycle, each of the network nodes will eventually perform this step. Furthermore, this is done in the order given by the tree from the SINK towards the leaves.

Whenever a node i receives a message MSG(d) from a neighbor ℓ, it estimates and stores its distance through this neighbor to the SINK. This distance is estimated as $d + d_{i\ell}$. As said before, the reevaluation of the estimated distance d_i is performed when receiving MSG from the preferred neighbor p_i. Node i calculates then the minimum of the estimated distances to the SINK through all those neighbors from which it has already received MSG (during the present update cycle). The node sets then d_i as this minimum. (Notice that d_i is only an "estimate" of the minimal distance to the SINK because it is sometimes calculated based upon part of the neighbors of i.)

When a node, say i, has received MSG(d) from all of its neighbors, it transmits MSG(d_i) to its p_i and then determines its new preferred neighbor p_i. This is done by choosing p_i as the neighbor which provides minimal estimated distance from i to the SINK. This choice is made among all neighbors of i and as such it may pick a neighbor different from the one which provided d_i (the calculation of the estimated distance d_i is usually based upon part of the neighbors). Since, as previously shown, each node i will eventually send MSG(d_i) to all its neighbors except p_i, the leaves of the directed tree will eventually receive MSG from all their neighbors. Thus they will send MSG to their preferred neighbor p_i and reevaluate a new p_i. It can be easily seen by induction, that each node will perform this step. This happens in the order given by the original directed tree, from the leaves towards the SINK.

Since the SINK denotes the destination, the SINK has no preferred neighbor, and therefore the SINK does not update p_{SINK} when it receives MSG(d) from all its neighbors. Instead, this event notifies the SINK that the update cycle has been properly completed. The SINK is not allowed to start a new update cycle until the previous cycle has been properly completed.

A node i always updates its preferred neighbor p_i to point towards a node j having estimated distance $d_j < d_i$. As proved in Section 3, this fact insures that the updated directed graph will remain a tree at any time.

The basic protocol can be formally defined by the basic algorithm performed by each node i. The latter is shown in Table 1 with the aid of a Finite State Machine. Node i can be in either of two states. It will be in state S2 after having received MSG from its preferred neighbor p_i and until it receives messages from all its neighbors. Otherwise node i will be in S1. The variables $D_i(\ell)$, one for each neighbor ℓ of i, store the values of the estimated distance through link ℓ to the SINK. The variables $N_i(\ell)$, one for each neighbor ℓ of i, denote flags which can take the value "RCVD" to mean that MSG(d) was received from link (i,ℓ) during the current cycle, or the value "nil" otherwise. CT is a control flag which can take over the values 0 or 1. We assume that when MSG(d) arrives from link ℓ, it is given to the algorithm in the format $MSG(d,\ell)$.

When $MSG(d,\ell)$ is processed, the flag $N_i(d)$ is set to RCVD, $D_i(\ell)$ is calculated, CT is set to 0, and then the Finite State Machine executes transitions until no more transitions are possible. Transition T12 can be executed if node i is in state S1 and Condition 12 is satisfied, i.e. the algorithm is processing a $MSG(d,\ell)$ in which $\ell = p_i$ and $CT = 0$. If T12 is executed, then node i goes to state S2 and Action 12 is performed, i.e. the estimated distance d_i is reevaluated and $MSG(d_i)$ is transmitted to each neighbor of i except the preferred neighbor p_i. In a similar way, T21 is executed when node i is in state S2 and Condition 21 is satisfied, in which case node i goes to state S1 and Action 21 is performed. The role of CT is to insure that T12 cannot be executed immediatly after T21 (for example, suppose node i is in state S1 and $MSG(d,\ell = p_i)$ arrives after messages have arrived for all other links of i. In this case, without CT, the sequence of transitions T12, T21, T12 will be performed in contradiction with the protocol). Notice that the sequence T12, T21 is permitted.

The use of the Finite State Machine for describing the relatively simple basic algorithm may appear superfluous. Its importance will become apparent when describing the more complex protocols and the proofs of their properties.

2.2 Handling Failures of Links

At our level of abstraction, the outage of a link is called "link failure". Transient (or transmission) failures can be masked out by the link protocol, and we are not concerned with them. If a link of the directed tree fails, then all the nodes which are predecessors of this link on the directed tree lose their route to the SINK, but they are unaware of this fact at the time of the failure. For example, if link (7,8) of Fig. 1 fails, nodes 6, 7 and 9 lose their route. Furthermore, if an update cycle is started, node 7 will not be able to receive MSG($d,\ell=8$) and therefore node 7, as well as nodes 6 and 9 will not be able to perform T12. In such a case we would like to recover by finding an alternative route (e.g. through node 5), but since the basic protocol allows changing estimated distance d_i and preferred neighbor p_i only after performing T12, there is need to provide an extension to handle this situation. Two actions must be taken by the extended protocol. First to inform nodes 7, 6 and 9 not to wait for triggering messages from the tree (and also that the existing tree has no meaning for them anymore) and second, to allow those nodes to choose their p_i whenever control messages from new cycles arrive. These features are in the sequel.

Whenever a node i discovers a failure of its link to the preferred neighbour p_i, it sets p_i = nil and $d_i = \infty$ to mean that its estimated distance to the SINK has become infinite. Then node i generates a special message MSG(∞) which propagates backwards through the tree to the nodes that lost their route, causing them also to set their best link to nil and the estimated distance to infinite. The propagation backwards is done as follows. Node i sends MSG(∞) to all its neighbors except p_i; if a node j receives MSG(∞) from a link other than p_j, it stores it but no other action is taken; if a node j receives MSG(∞) from p_j, then it transmits MSG(∞) to all its neighbors except p_j and sets p_j = nil, $d_j = \infty$. When a node establishes p_i = nil, $d_i = \infty$, it is said to enter state S3 (see Table 3).

The second part of the recovery, called "reattachment", consists of choosing a new best link by those nodes i having p_i = nil. The reattachment takes place if one of the following two situations occurs. One possibility is that a node with p_i = nil receives on one of its links, ℓ say, a message MSG($d \neq \infty$) and the node is assured that this message was generated by an update cycle that started <u>after the failure</u> that caused p_i = nil. A second possibility is that at the time p_i is set to nil, such a message has already been received at node i. The reattachment consists of setting $p_i = \ell$, going to state S2 and effecting the same operations as in T12. This, together with other mechanisms to be described, guarantees that if a failure (or multiple failures) occurs, and if indeed a new update cycle is started, all nodes physically connected to the SINK will eventually belong to a non-disrupted directed tree rooted at the SINK.

As mentioned above, there is need to guarantee that reattachment is performed only as a result of receiving a message generated by an update cycle which started after the failure. This can be achieved by numbering the update cycles with nondecreasing numbers as described below. Each node i will have a counter number n_i which denotes the cycle number currently handled by this node, and all messages transmitted by i will carry n_i in addition to d_i, i.e. they will be MSG(n_i,d_i). The SINK may increase its n_{SINK} before starting a new update cycle, as explained later.

A node i receiving MSG(m,d) on its p_i updates its n_i to equal m. Now, reattachment is done by a node i with p_i = nil if an MSG(m,d) with $m > n_i$ is received (or was previously received).

When an MSG(m,d) is received from link ℓ by node i, in addition of storing d in $D_i(\ell)$, there is need to remember also the value of m. This can be saved in $N_i(\ell)$, which can now take the values nil,0,1,2,3,...; instead of nil and RCVD as in the basic algorithm.

If a failure occurs in a link not belonging to the directed tree, no route is disrupted. However, if this link is connected to a node in state S2, it is convenient to prevent T21 from happening at this node for this update cycle. This will avoid nodes to update routes based upon information which is invalidated by the failure and, more important, will preclude proper completion from happening. Thus, proper completion will indicate to the SINK that the update cycle was completed without failures interfering with the process. Prevention of T21 is accomplished by introducing an additional state, $S\tilde{2}$, into which a node enters if a nonpreferred link fails while the node is in S2. A node i will leave $S\tilde{2}$ whenever new information is received on p_i (see Table 3).

The described protocol allows the SINK to behave as follows. If an update cycle started with $n_{SINK} = m$ completes properly, the SINK is allowed to start the next update cycle with the same n_{SINK}. On the other hand, the SINK may at any time increase n_{SINK} and start a new update cycle with an n_{SINK} larger than those used before, even if previous cycles have not been properly completed. (Notice that in any case the values of n_{SINK} are non-decreasing with time.) As proved later, if a new update cycle is started while increasing n_{SINK}, it will eventually "cover" all previous cycles. Also, if failures do not occur for a long enough time, the new cycle will be properly completed, and all failures will be recovered, i.e. for all nodes i physically connected to the SINK, the directed graph of (i,p_i) will form a tree rooted at the SINK.

Table 2 summarizes the variables used by the algorithm performed by an arbitrary node i as its part of the protocol. $F_i(\ell)$ denotes the status of link ℓ as considered by node i, i.e. $F_i(\ell) = UP$ if ℓ is considered operational and $F_i(\ell) = DOWN$ if ℓ is considered unoperational. $F_i(\ell)$ can take also the value "READY", whose use will be described when dealing with the problem of links becoming operational. At that time, the role of $z_i(\ell)$ will also become clear. The variable mx_i stores the value of the largest update cycle number m of all the messages $MSG(m,d,\ell)$ received by node i. The rest of the variables and their use were already described. The local link protocols controlling the operations of the links connected to node i may relay to the algorithm performed by node i four types of messages, and they are als summarized in Table 2. MSG denotes an updating message, FAIL(ℓ) denotes the detection of the failure of link ℓ, and the remaining two will be described later. The exact properties required from the local protocol to insure proper operation of the network protocol will be discussed in Section 2.7.

Table 3 describes the generalized algorithm of node i for the protocol which handles topological changes. The protocol as described up to now is implemented by the algorithm of Table 3 if ignoring steps I.1, I.2.4, I.3.1, I.4, II.1.5, II.2.5 and II.7.7. These steps relate mainly to links becoming operational and will be discussed in subsequent sections. Table 3 uses a notation similar to the one of Table 1. States S1, S2 and transitions T12 a and T21 are similar to those described in Table 1 for the basic algorithm. State S3 denotes the situation where the node has p_i = nil, which results from receiving a FAIL or a MSG with $d = \infty$ from p_i. State $\tilde{S2}$ denotes a state similar to S2, but from which a transition T21 is precluded. As previously described, the algorithm goes to such a state $\tilde{S2}$ if while at S2 a failure is detected from a link other than p_i. The "Facts" given in the algorithm are displayed for helping in its understanding and are proven in Theorem 2 of Section 3. A Fact holds if the transition under which it appears is performed.

2.3 Starting a New Update Cycle

There exist several procedures for starting a new update cycle and setting the corresponding n_{SINK} in a way which satisfy the required behaviour from the SINK as described in Section 2.2. Two of these procedures are described next.

Version 1: At given intervals of time, or as a result of the detection of a change in the traffic pattern, the SINK increments n_{SINK} and starts a new update cycle. The above version may make use of a time-out to trigger a new update cycle if the previous one is not properly completed within certain time. If a failure occurs after proper completion, there is no direct triggering of a new update cycle, and thus recovery can be achieved only whenever the SINK decides to start a new update cycle. In addition, this version unnecessarily increments n_{SINK} for every update; hence an unnecessarily large number of bits to represent n_{SINK} is required. These two disadvantages are overcome by the next version.

Version 2: In order to cope with changes in traffic patterns, after proper completion of the previous update cycle, the SINK may start a new update cycle with the same n_{SINK}. In addition, whenever a node i detects a failure of a link attached to it, the node generates a special message $REQ(n_i)$ which is forwarded through the directed path of preferred links to the SINK. If such a $REQ(m)$ arrives at a node i having p_i = nil, the REQ is discarded. In Section 3 it is shown that if a $REQ(m1)$ is generated and forwarded as mentioned above, then some $REQ(m2)$, $m2 \geq m1$ will actually arrive at the SINK, within finite time. Whenever a $REQ(m)$ arrives at the SINK, and if $m = n_{SINK}$, then n_{SINK} is incremented and a new update cycle is started. This cycle will take care of recovery from the failure that generated the REQ(m). If $m < n_{SINK}$ such a cycle was already started and the REQ(m) can simply be ignored. (Notice that m cannot be larger than n_{SINK}.) This version guarantees that if an update cycle with $n_{SINK} = m$ is started, the cycle will be properly completed in finite time or else, a failure has occurred and a REQ(m) will arrive at the SINK. (This is proved in Section 3.) Thus, there is no need for a time-out to make sure that the SINK will not wait indefinitely for the proper completion of an update cycle. The additions to the algorithm for implementing this version are given in I.1 and I.2.4 of Table 3. In the rest of the paper, we assume that this version is implemented, although most of the results are also applicable to Version 1.

2.4 Handling Links Becoming Operational

If link (i,ℓ) is down, i.e. $F_i(\ell) = F_\ell(i) =$ DOWN, and it becomes operational, nodes i and ℓ should coordinate the operations necessary to bring the link up. Otherwise, a deadlock could occur, for instance, if i sets $F_i(\ell) =$ UP while at S2 and ℓ sets $F_\ell(i) =$ UP after performing T21 of the same update cycle. In this case, i will not perform T21 until receiving a message from ℓ, and such a message will not be sent because ℓ already completed this update cycle, i.e. deadlock.

The coordination is achieved by having both nodes bring the link up just before starting to perform their part of the <u>same new</u> cycle. This is done in two steps. First, i and ℓ compare n_i and n_ℓ via the local protocol and decide to bring up the link when starting to process the first cycle with number strictly higher than $\max(n_i,n_\ell)$. This fact is remembered at the nodes by setting $z_i(\ell)$ and $z_\ell(i)$ to $\max(n_i,n_\ell)$, as well as $F_i(\ell)$ and $F_\ell(i)$ to "READY". In addition, $N_i(\ell)$ and $N_\ell(i)$ are set to nil and $REQ(z_i(\ell))$ is generated by nodes i and ℓ and forwarded to the SINK in the same way as described in Section 2.3 (Version 2) for failures. This will guarantee that an update cycle with n_{SINK} larger than $z_i(\ell)$ (and $z_\ell(i)$) will be started. This first step of the coordination at node i is done by message WAKE(ℓ) given by the local protocol to the algorithm. The actions performed by the algorithm when receiving such a message are described in I.4 of Table 3. The synchronization assumes that the execution of WAKE(ℓ) and WAKE(i) are simultaneously started at nodes i and ℓ respectively, in order to guarantee that $z_i(\ell) = z_\ell(i)$. However, it may happen that a failure occurs again in the link and one of the nodes succeeds to complete the synchronization while the other node does not. The protocol allows for such a situation and only requires that the link protocol ends the synchronization (successfully or unsuccessfully) within finite time. If the synchronization is unsuccessful, no action is taken by the node, and the link will remain DOWN from this node's point of view. Section 2.7 gives a more formal and complete list of the requirements that the link protocol should satisfy.

The second step of bringing the link (i,ℓ) up is done by node i (i.e. $F_i(\ell)$ is set from READY to UP) when node i receives MSG from link ℓ or when the node counter number n_i becomes larger than $z_i(\ell)$. This is represented respectively by I.3.1 and II.1.5, II.2.5, II.7.7, of Table 3.

2.5 The Algorithm for the SINK

The algorithm for the SINK, assuming that Version 2 of Section 2.3 is chosen, appears in Table 4. Most of the algorithm was already informally discussed in previous sections. The main difference between the algorithm for the SINK and that for an arbitrary node i is that the first does not need to keep the following variables:

- p_i (which is not defined for the SINK)
- d_i (which is always 0 for the SINK)
- $D_i(\ell)$ (which is only needed to update d_i and p_i)
- mx_i (n_{SINK} is always the largest update number)
- $z_i(\ell)$ (during WAKE synchronization, $z_{SINK}(\ell)$ is always set to $n_{SINK} = \max(n_{SINK}, n_\ell)$)

In addition, the algorithm may receive a "START" message from the "outside world" which will cause it to start a new cycle, provided that the last one was properly completed. WAKE and REQ call also for the execution of the Finite-State-Machine, and as a result WAKE as well as $REQ(m = n_{SINK})$ will cause an increment of n_{SINK} and a new update cycle will be started. States S1 and S2 are similar to the corresponding states of the algorithm for an arbitrary node i. However, S1 means for the SINK that the last update cycle was properly completed, and S2 means that the current update cycle is not yet completed. T12 and T22 represent the starting of a new update cycle and T21 the proper completion. For the SINK there is no need for states equivalent to S3 and $\tilde{S2}$.

2.6 Initialization of the Protocol

Any arbitrary node i comes into operation in state S3, with node counter number $n_i = 0$, preferred neighbor p_i = nil, and $F_i(k)$ = DOWN for all k. The value of the remaining variables is immaterial. From this initial condition, the local protocol may try to wake the links and it proceeds operating as defined by the algorithm (Table 3). The SINK comes into operation in state S1, with $n_{SINK} = 0$ and $F_i(k)$ = DOWN for all k, and proceeds in the same way but according to the algorithm of Table 4.

2.7 Properties Required from the Local Protocol

On each link of the network there is a local protocol that is in charge of exchanging messages between neighbors. Our main algorithm assumes that the following properties hold for the local protocol:

2.7.1 All links are bidirectional (duplex).

2.7.2 $d_{i\ell} > 0$ for all links (i,ℓ) at all times.

2.7.3 If a message is sent by node i to a neighbor ℓ, then in finite time, either the message will be received correctly at ℓ or $F_i(\ell) = F_\ell(i) = \text{DOWN}$. Observe that this assumption does not preclude transmission errors that are recovered by the local protocol (e.g. "resend and acknowledgement").

2.7.4 Failure of a node is considered as failure of all links connected to it.

2.7.5 A node i comes up in state S3, with $n_i = 0$, $p_i = \text{nil}$, and $F_i(\ell) = \text{DOWN}$ for all links (i,ℓ).

2.7.6 The processor at node i receives messages from link (i,ℓ) on a first-in-first-served (FIFO) basis.

2.7.7 A link (i,ℓ) is said to have become operational as soon as the local protocol discovers that the link can be used. Links (i,ℓ) and (ℓ,i) become operational at the same time and subject to the following restrictions, a WAKE "message" is delivered in this case to each of the processors i and ℓ.

WAKE(ℓ) can be received at node i only if

(a) node ℓ receives WAKE(i) at the same (virtual) time;

(b) there are no other outstanding messages on link (i,ℓ) and on (ℓ,i);

(c) $F_i(\ell) = F_\ell(i) = \text{DOWN}$.

2.7.8 If $F_i(\ell)$ = DOWN, the only message that the processor at i can receive from ℓ is WAKE(ℓ).

2.7.9 (a) If $F_i(\ell) \neq$ DOWN and $F_\ell(i) \neq$ DOWN and $F_i(\ell)$ goes to DOWN, then $F_\ell(i)$ goes to DOWN in finite time.

(b) If $F_i(\ell) = F_\ell(i)$ = DOWN and $F_i(\ell)$ goes to READY, then in finite time, either $F_\ell(i)$ goes to READY or $F_i(\ell) = F_\ell(i)$ = DOWN.

2.7.10 When two nodes i and ℓ receive WAKE as described in 2.7.7, a "synchronization" between i and ℓ is attempted. At either end the synchronization may or may not be successful (the latter because of a new failure). If it is successful, the node proceeds as in Step I.4 of Table 3. If not, then no action is taken.

3. PROPERTIES AND VALIDATION OF THE ALGORITHM

Some of the properties of the algorithm have already been indicated in previous sections. Here we state them explicitly along with some of the others. We start with properties that hold throughout the operation of the network, some of them referring to the entire network at a given instant of time and some to a given node or link as time progresses. Then recovery of the network after topological changes is proved through a series of theorems, and finally we state and prove the fact that the algorithm achieves shortest weighted routes. We may point out, that the most important features of the algorithm are given in Theorems 1, 4, 5 and 6.

Before stating the main properties of the algorithm, we need several definitions and notations:

S1, S2, $S\tilde{2}$, S3 = states of the Finite-State Machine.

PC(m) = time of proper completion with cycle counter number m.

S1[n] = state S1 with node counter number $n_i = n$, and similarly for S2[n], S3[n], $S\tilde{2}$[n].

Whenever we want to refer to a quantity at a given time t, we add the time in in parentheses (e.g. $p_i(t)$ means preferred neighbor p_i of node i at time t, $N_i(\ell)(t)$ means variable $N_i(\ell)$ at time t, etc.)

$s_i(t)$ = state and possibly node counter number n_i of node i at time t. Therefore we sometimes write $s_i(t)$ = S3 for instance, and sometimes $s_i(t)$ = S3[n].

We use a compact notation to describe changes accompanying a transition, as follows:

Txy[t,i,MSG(m1,d1,ℓ1),SEND(m2,d2,ℓ2),(n1,n2),(d1,d2),(p1,p2),(mx1,mx2)] (1a)

will mean that transition from state Sx to state Sy takes place at time t at node i caused by receiving MSG(m1,d1) from neighbor ℓ1; in this transition i sends MSG(m2,d2) to ℓ2, changes its node counter number

n_i from n1 to n2, its estimated distance to destination d_i from d1 to d2, its preferred neighbor p_i from p1 to p2 and the largest update counter number received up to now mx_i from mx1 to mx2. Similarly,

$$Txy[t,i,FAIL(\ell),SEND(m2,d2,\ell 2),(n1,n2)(d1,d2),(p1,p2),(mx1,mx2)] \tag{1b}$$

denotes the same transition as above, except that it is caused by receiving FAIL(ℓ) from neighbor ℓ. For simplicity, we delete all arguments that are of no interest in a given description, and if for example n1 is arbitrary we write $(\phi,n2)$ instead of (n1,n2). Similarly, if one of the states is arbitrary, ϕ will replace this state. In particular observe that

$$T\phi 2[t,SINK,(\phi,n2)] \tag{2}$$

means that an updating cycle with number n2 is started at time t and

$$T21[t,SINK,(n2,n2)] \tag{3}$$

means that proper completion of the cycle occurs at time t. If Txy[t], then we use the notations:

t- = time just before the transition,

t+ = time just after the transition.

We also use

$$[t,i,MSG(m,d,\ell)] \tag{4}$$

to denote the fact that a message MSG(m,d) is received at time t at i from ℓ, whether or not the receipt of the message causes a transition.

Finally, at a given instant t, we define the <u>Routing Graph</u> RG(t) as the directed graph whose nodes are the network nodes and whose arcs are given by the preferred neighbors p_i, namely there is an arc from node i to node ℓ if and only if $p_i(t) = \ell$. For example, the routing graph of the network in Fig. 1a is given in Fig. 1b. In order to describe properties of the RG(t), we also define an order for the states by $S3 > S2 = \tilde{S2} > S1$.

Also, if Sx and Sy are states, then the notation $Sx \geq Sy$ means $Sx > Sy$ or $Sx = Sy$. For conceptual purposes, we regard all the actions associated with a transition of the Finite-State Machine to take place at the time of the transition.

Theorem 1

At any instant of time, RG(t) consists of a set of disjoint trees wi h the following ordering properties:

i) the roots of the trees are the SINK and all nodes in S3;

ii) if $p_i(t) = \ell$, then $n_\ell(t) \geq n_i(t)$;

iii) if $p_i(t) = \ell$ and $n_\ell(t) = n_i(t)$, then $s_\ell(t) \geq s_i(t)$;

iv) if $p_i(t) = \ell$ and $n_\ell(t) = n_i(t)$ and $s_\ell(t) = s_i(t) = S1$, then $d_\ell(t) < d_i(t)$.

The proof of Theorem 1 is given in Appendix A. According to it, the RG consists at any time of a set of disjoint trees, i.e. it contains no loops. Observe that a tree consisting of a single isolated node is possible. The algorithm maintains a certain ordering in a tree, namely that concatenation of (n_i, s_i) is nondecreasing when moving from the leaves to the root of a tree and in addition, for nodes in S1 and with the same node counter number, the estimated distances d_i to the SINK are strictly decreasing.

In addition to properties of the entire network at each instant of time, we can look at local properties as time progresses. Some of the most important are given in the following theorem whose proof appears in Appendix A (see c) and d) in Theorem A.1).

Theorem 2

i) For a given node i, the node counter number n_i is nondecreasing and the messages $MSG(m,d)$ received from a given neighbor have non-decreasing numbers m.

ii) Between two successive proper completions $PC(\bar{m})$ and $PC(\bar{\bar{m}})$, for each given m with $\bar{m} \le m \le \bar{\bar{m}}$, each node sends to each of its neighbors at most one message $MSG(m,d)$ with $d < \infty$.

iii) Between two successive proper completions $PC(\bar{m})$ and $PC(\bar{\bar{m}})$, for each given m with $\bar{m} \le m \le \bar{\bar{m}}$, a node enters each of the sets of states $\{S1[m]\}$, $\{S2[m], \tilde{S2}[m]\}$, $\{S3[m]\}$ at most once.

iv) All "Facts" in the formal description of the algorithm in Section II are correct.

A third theorem describes the situation in the network at the time proper completion occurs:

Theorem 3

At $PC(\bar{m})$, the following hold for each node i:

i) If $n_i = \bar{m}$, then $s_i = S1$ or $s_i = S3$.

ii) If a message $MSG(\bar{m},d)$ with $d \neq \infty$ is on its way to i, then $s_i = S3$ and $n_i = \bar{m}$.

iii) If either ($n_i = \bar{m}$ and $s_i = S1$) or $n_i < \bar{m}$, then for all k such that $F_i(k) = UP$, it cannot happen that $\{N_i(k) = \bar{m}, D_i(k) < \infty\}$.

A combined proof is necessary to show that the properties appearing in Theorems 1, 2, 3 hold. The proof uses a two-level induction, first assuming properties at PC to hold, then showing that the other properties hold between this and the next PC and finally proving that the necessary properties hold at the next PC. The second induction level proves the properties between successive proper completions by assuming that the property holds until just before the current time t and then showing that any possible change at time t preserves the property. The entire rigorous procedure appears in Appendix A.

In order to introduce properties of the algorithm regarding normal activity and recovery of the network, we need several definitions.

Definition

We say that a link (i,ℓ) is potentially working if $F_i(\ell) \neq$ DOWN and $F_\ell(i) \neq$ DOWN, and a link (i,ℓ) is working if $F_i(\ell) = F_\ell(i) =$ UP. Two nodes in the network are said to be potentially connected at time t if there is a sequence of links that are potentially working at time t connecting the two nodes. A set of nodes is said to be strongly connected to the SINK if all nodes in the set are potentially connected to the SINK and for all links (i,ℓ) connecting those nodes, we have either $F_i(\ell) = F_\ell(i) =$ UP or $F_i(\ell) = F_\ell(i) =$ DOWN.

Definition

Consider a given time t, and let ml be the highest counter number of cycles started before t. We say that a pertinent topological change happens at time t if the algorithm at a node i with $n_i(t-) = ml$ receives at time t a message WAKE(ℓ) resulting in successful WAKE synchronization or a message FAIL(ℓ). Observe from steps I.2 and I.4 of Table 3 that REQ(ml) is generated and sent if and only if a pertinent topological change happens at a node i with $p_i \neq$ nil. Also note, that a pertinent topological change happens if and only if node i has a link (i,k) such that at time t, $F_i(k)$ changes from DOWN to READY or from either UP or READY to DOWN (see Fig. 2).

<u>Theorem 4</u> (Normal activity)

Let

$L(t)$ = {nodes potentially connected to SINK at time t} ,

$H(t)$ = {nodes strongly connected to SINK at time t} .

Suppose

$$T\phi2[t1,SINK,(m1,m1)], \tag{5}$$

namely a cycle is started at t1 with a number that was previously used. Suppose also that no pertinent topological changes have happened while $n_{SINK} = m1$ before t1 and no such changes happen for long enough time after t1. Then there exist t0, t2, t3 with $t0 < t1 < t2 < t3 < \infty$ such that a), b), c), d) hold:

a) $T21[t0,SINK,(m1,m1)];$ (6)

b) $\forall t$ in the interval $[t0,t3]$, we have $H(t) = L(t) = L(t0)$;

c) for all $i \in L(t0)$, we have

$$T\phi2[t2_i,i,(m1,m1)] \tag{7}$$

for some time $t2_i$ in the interval $[t1,t2]$;

d) i) $T21[t3,SINK,(m1,m1)]$; (8)

ii) $RG(t3)$ for all nodes in $L(t0)$ is a single tree rooted at SINK.

In words, Theorem 4 says that under the given conditions, if a new cycle starts with a number that was previously used, then Proper Completion with the same number has previously occurred and the new cycle will be properly completed in finite time while connecting all nodes of interest (i.e. in $L(t0)$) to the SINK, both strongly and routingwise. The proof of Theorem 4 is given in Appendix B.

The recovery properties of the algorithm are described in Propositions 1, 2 and in Theorem 5. The proofs of the propositions appear in Appendix B.

Proposition 1

Let $L(t)$, $H(t)$ be as in Theorem 4. Suppose

$$T\phi 2[t1,SINK,(m1,m2)] \; ; \quad m2 > m1 , \tag{9}$$

namely a cycle starts at time t1 with a number that was not previously used. Suppose also that no pertinent topological changes happen for a long enough period after t1. Then

a) there exists a time t2, with $t1 \leq t2 < \infty$, such that

i) for all $i \in L(t2)$

$$T\phi 2[t2_i,i,(\phi,m2)] \tag{10}$$

happen at some time $t2_i$ with $t1 \leq t2_i \leq t2$.

ii) $H(t2) = L(t2)$.

b) There exists a time $t3 < \infty$ such that

i)
$$T21[t3,SINK,(m2,m2)] \; ; \tag{11}$$

ii) $\forall t$ in the interval $[t2,t3]$, we have $H(t) = L(t) = H(t2)$;

iii) RG(t3) for all nodes in L(t3) is a single tree rooted at SINK.

Part a) of Proposition 1 says that under the stated conditions, all nodes in L(t) will eventually enter state S2[m2]. Part b) says that the cycle will be properly completed and all nodes potentially connected to the SINK at time PC(m2) will actually be strongly connected to the SINK and will also have a routing path to the SINK.

Finally, we observe that reattachment of a node loosing its path to the SINK or bringing a link up requires a cycle with a counter number higher than the one the node currently has. Proposition 2 ensures that such a cycle has been or will be started in finite time by the SINK.

Proposition 2

Suppose a node i receives FAIL(ℓ) while $n_i = ml$ or a successful WAKE(ℓ) synchronization occurs at node i while $z_i(\ell) = ml$. Then the SINK has received before t a message REQ(ml) or will receive such a message in finite time after t.

Propositions 1 and 2 are combined in:

Theorem 5 (Recovery theorem)

Let $L(t)$, $H(t)$ be as in Theorem 4. Suppose there is a time $t1$ after which no pertinent topological changes happen in the network for long enough time. Then there exists a time $t3$ with $t1 \leq t3 < \infty$ such that all nodes in $L(t3)$ are strongly connected to the SINK and are on a single tree rooted at SINK.

Proof

Let $t0 \leq t1$ be the time of the last pertinent topological change before $t1$. Let i be the node detecting it and let $m = n_i(t0-)$. Then Proposition 2 assures that a message REQ(m) arrives at some finite time at SINK. Let $t2 < \infty$ be the time the first REQ(m) message arrives at SINK. Condition 12 or 22 in Table 4 dictates that SINK will start at time t2 a new cycle, with number $ml = m+1$. Since by the definition of pertinent change, m is the largest number at time $t0$, we have that $t0 < t2$. By assumption, no pertinent topological changes happen after time $t0$ for a long enough period, so that no such changes happen after time $t2$. Consequently Proposition 1 holds after this time and the assertions of the Theorem follows.

Theorem 6 (Shortest paths)

With the notations of Theorem 5, suppose the conditions of Theorem 5 hold and in addition, suppose that the weights $d_{i\ell}$ of the links are time invariant for a long enough period after $t1$. Then, after completion of a finite number of cycles after t3, the routing graph RG will provide the <u>shortest route in terms of the weights</u> $d_{i\ell}$ from each node in L(t3) to the SINK. Let SR be the graph providing the shortest routes in terms of $d_{i\ell}$. Then the necessary number of cycles is bounded from above by the largest distance from SINK in terms of number of hops on SR.

<u>Proof</u>

Observe from steps II.1.3 and II.3.7 in Table 3, that during the first cycle after t3 all nodes closest to SINK on SR will have p_i = SINK and will never change p_i afterwards.

Next, consider any connected subgraph A of SR that includes the SINK. Suppose that at the time of a cycle completion SR and RG coincide for nodes in A. Then these nodes will never change their preferred neighbors p_i afterwards. Also during the next cycle at least the nodes neighboring A on SR will change their p_i such that RG and SR will coincide for them too, and this proves the assertion.

IV. DISCUSSION AND CONCLUSIONS

The paper presents an algorithm for constructing and maintaining loop-free routing tables in a data-network, when arbitrary failures and additions happen in the network. Clearly, the properties that are rigorously proved in Section 3 and the Appendices hold also for several other versions of the algorithm, some of them simpler and some of them more involved than the present one. We have decided on the present form of the algorithm as a compromise between simplicity and still keeping some properties that are intuitively appealing. For example, one possibility is to increase the update cycle number every time a new cycle is started. This will not simplify the algorithm, but will greatly simplify the proofs. On the other hand, it will require many more bits for the update cycle and node numbers m and n_i than the algorithm given in the paper. Another version of the algorithm previously considered by us was to require that every time a node receives a number higher than n_i from some neighbor, it will "forget" all its previous information and will "reattach" to that node immediately, by a similar operation to transition T32. This change in the algorithm would considerably simplify both the algorithm and the proofs, but every topological change will affect the entire network, since after any topological change, all nodes will act as if they had no previous information. On the other hand, the version given in the paper "localizes" failures in the sense that only those nodes whose best path to SINK was destroyed will have to forget all their previous information. This is performed in the algorithm by requiring that nodes not in S3 will wait for a signal from the preferred neighbor p_i before they proceed, even if they receive a number higher than n_i from other neighbors. The signal may be either ∞, in which case the node enters S3 (and eventually reattaches) or less than ∞, in which case the node proceeds as usual.

A final remark regarding the amount of control information required by the protocol. Observe that since for each update and for each destination each node sends over each link the distance d_i and the node counter number n_i, the amount of information sent over each link is of the same order of magnitude as the ARPA routing protocol [7]. The difference is that the latter allows information for all destinations to be sent in one message, whereas our protocol requires in principle separate messages for different destinations (although sometimes several messages may be packed together). If the overhead for control messages is not too large however, the extra load will not be significant.

Appendix A

We organize the proofs as follows: We start with the statements of a few properties that follow immediately from the formal description of the algorithm in Table 3. Lemmas A.1 - A.4 and Theorem A.1 contain the proofs of Theorems 1, 2 and 3, together with some other properties needed in the proofs themselves. Theorem 4 and Propositions 1 and 2 will be proved in Appendix B.

Properties of the Algorithm

R1 Any change in n_i, s_i, p_i, or sending any message (m,d) can happen only while i performs a transition.

R2 $Txy[t,i,SEND(m,d),(\phi,n2),(\phi,d2),(\phi,mx2)]$ implies $d = d2$.

If $d \neq \infty$, then

i) $Txy = T12$ or $T21$ or $T22$ or $T32$ or $\tilde{T}22$

ii) $n2 = mx2 = m$

If $d = \infty$, then

iii) $Txy = T13$ or $T23$ or $\tilde{T}23$

iv) $n2 = m$.

R3 $T32[t,i,(n1,n2)] \Rightarrow n2 > n1$

R4 $s_i(t) = S3 \Leftrightarrow p_i(t) = nil \Leftrightarrow d_i(t) = \infty$

R5 $Txy[t,i,(p1,p2)]$, $p1 \neq nil$, $p2 \neq p1 \Rightarrow Txy = T13$ or $T21$ or $T23$ or $\tilde{T}23$.

R6 $mx_i(t)$ is nondecreasing in time for any i.

R7 In the Finite-State-Machine, no two conditions can hold at the same time. This implies that the order of checking the conditions of the transitions is irrelevant.

R8 For all t and all nodes i in the network, $n_{SINK}(t) \geq n_i(t)$ and $n_{SINK}(t) \geq mx_i(t)$.

R9 The Finite-State Machine has two types of transitions. The first type is effected directly by the incoming message, while the second type is caused by the situation in the memory of the node. Transitions T23 and T21 are of the second type, all others are of the first type. Each message can trigger only one transition of the first type, and this transition comes always before transitions of the second type. This is controlled by the variable CT in Table 3.

R10 The possible changes of $F_i(\ell)$ are given in Fig. 2. The types of messages causing them are also shown. A pertinent topological change happens iff $F_i(\ell) \leftarrow$ DOWN or $F_i(\ell)$ changes from DOWN to READY at a node i with $n_i(t-) = m1$, where $m1$ is the highest counter number of cycles started before t.

The following lemma says that the node number n_i can be changed only when receiving a message from the preferred neighbor p_i and then, the new number is exactly the cycle number m received in that message. It also gives conditions for leaving state S3.

<u>Lemma A.1</u>

If

$$\mathrm{Txy}[t, i, \mathrm{MSG}(m,d,\ell), (n1,n2), (p1,\phi)] \tag{A.1}$$

or

$$\mathrm{Txy}[t,i, \mathrm{FAIL}(\ell),(n1,n2),(p1,\phi)]$$

then

a) $p1 \neq nil,\ n2 \neq n1 \Rightarrow \ell = p1$ and $n2 = m$;

b) $p1 = nil \Rightarrow n2 > n1$, and also $\exists k$ s.t. $F_i(k)(t-) = \mathrm{UP},\ N_i(k)(t-) = n2$.

Proof

a) From the algorithm we see that T21, T22, $\tilde{T}22$ do not apply here since they imply n2 = n1. Also T32 does not apply, since then p1 = nil. If T13, T23 or $\tilde{T}23$ is caused by FAIL(ℓ) then n2 = n1, so this case does not apply either. In all other cases, n2 = m and p1 = ℓ (see II.1.4, II.2.1, II.2.4 in Table 3).

b) p1 = nil implies Txy = T32 and the assertion follows from steps II.7.1 and II.7.5 in Table 3.

The next lemma proves statement i) of Theorem 2 and shows the role of the node counter number n_i. Here we see for the first time that several properties have to be proved in a common induction.

Lemma A.2

a) [i,t1,MSG(m1,d1,ℓ)], [i,t2,MSG(m2,d2,ℓ)], $\quad$ t2 > t1 $\Rightarrow$ m2 $\geq$ m1.

b) $T\phi\phi$[t,i,(n1,n2)] $\Rightarrow$ n2 $\geq$ n1 .

c) Let $M_i(t,p_i(t))$ denote the counter number m of the last message MSG(m,d) received at i before or at time t from the preferred neighbor $p_i(t)$. Then

$$n_i(t) \leq M_i(t,p_i(t)) \tag{A.2}$$

Proof

The proof proceeds by induction. We assume that a), b), c) hold up to, but not including, time t <u>for all nodes in the network</u>. We then prove below that any possible event at time t preserves the properties. This, combined with the fact that a), b), c) hold trivially at the time any node comes up for the first time, completes the proof.

a) Suppose $t = t2$. Then by FIFO and property R2, $\exists t3,t4$ with $t3 < t4 < t$ such that $n_\ell(t3) = m1$ and $n_\ell(t4) = m2$. By induction hypothesis on b), n_ℓ was nondecreasing up to (but not including) time t, so $m1 \le m2$.

b) Observe first from steps II.2.4 and II.5.1 in Table 3,

$$T\phi\phi[t,i,FAIL(\ell),(n1,n2)]$$

implies $n2 = n1$, so that the statement is true in this case. We therefore have to check only the case when the transition is caused by MSG. Suppose

$$T\phi\phi[t, i, MSG(m,d,\ell),(n1,n2),(p1,p2)] \tag{A.3}$$

happens. If $n2 = n1$, q.e.d. If $n2 \neq n1$, then Lemma A.1 implies that either $p1 = nil$ <u>or</u> $(p1 = \ell,\ n2 = m)$. If $p1 = nil$, q.e.d. from Lemma A.1. If $(p1 = \ell,\ n2 = m)$,

then

$$n1 \le M_i(t-,p1) = M_i(t-,\ell) \le M_i(t,\ell) = m = n2 \tag{A.4}$$

where the inequalities follow respectively from induction hypothesis on c) and from applying a) at time t.

c) We have to show that if

$$[i, t, MSG(m,d,\ell),(n1,n2),(p1,p2)] \tag{A.5}$$

then

i) $\ell = p1 = p2$ implies $n2 \le m$, and

ii) $p2 \neq p1$, $p2 \neq nil$ implies $n2 \le M_i(t+,p2)$.

To do this we check all possible transitions and also the case when the received message causes no transition. T13, T23 and $\tilde{T23}$ do not apply here because then $p1 \neq nil$, $p2 = nil$. If $\tilde{T22}$ or no transition, then $p2 = p1$ and $n2 = n1$, and we have

$$n2 = n1 \le M_i(t-,p1) \le M_i(t+,p1) = M_i(t+,p2) = m \ ,\tag{A.6}$$

where the inequalities follow from the induction hypothesis and from a) respectively. For the other transitions we have

T12, T22 and $\tilde{T}22$ imply $\ell = p1 = p2$, $n2 = m$ (see II.1.1 and II.1.4 in Table 3).

T21 implies $p2 \neq nil$, and then the counter number of the last message received from any neighbor before $t+$ is $n1 = n2 = m$.

T32 implies $p2 \neq p1$, $p2 \neq nil$ and then from steps II.7.4, II.7.5 II.7.1 in Table 3, $n2 = mx_i(t-)$, $p2 = k^*$, $M_i(t+,k^*) = N_i(k^*)(t-) = mx_i(t-)$.

The next lemma shows what are messages that can travel on a line after a failure or after a message with $d = \infty$.

<u>Lemma A.3</u>

a) If

$$[i,t1,MSG(m1,d1,\ell)],\ [i,t2,MSG(m2,d2,\ell)] \qquad (A.7a)$$

where $t2 > t1$, $d1 = \infty$, then $m2 > m1$.

b) If

$$[i,t1,FAIL(\ell)],\ [i,t2,MSG(m2,d2,\ell)] \qquad (A.7b)$$

where $t2 > t1$, then $m2 > n_i(t1)$ and also $m2 > n_\ell(t1)$.

<u>Proof</u>

a) $\exists t3 < t1$ such that

$$T\phi3[\ell,t3,SEND(m1,d1,i),(\phi,n2)] \qquad (A.8)$$

and from property R2 we have $m1 = n2$. The next transition of ℓ must be

$$T32[\ell,(n2,n3)],\ n3 > n2$$

so that by Lemma A.2 b) and R2, node ℓ will never send after t3 any message $MSG(m,d)$ with $m \leq m1$. FIFO at node i completes the proof.

b) After failure, a link (i,ℓ) can be brought up only with numbers strictly higher than $z_i(\ell)$ as defined in step I.4 of Table 3. Since n_i is non-decreasing by Lemma A.2. b), the proof is complete.

Lemma A.4

If $F_i(\ell)(t)$ = READY and

$$[t,i,\mathrm{MSG}(m,d,\ell)] \tag{A.9}$$

then $m > z_i(\ell)(t)$. Observe that this is Fact I.3.1 in Table 3.

Proof

From steps I.1-I.4 in Table 3 and property 2.7.7 in Sec. 2.7, $F_i(\ell)$ can go to READY only from DOWN and only when successful synchronization of WAKE(ℓ) occurs at i. Let $t1 < t$ be the time this occurs. By property 2.7.7, at time $t1$ there are no outstanding messages on (i,ℓ) or (ℓ,i) and $z_i(\ell)$ is established as $\max\{n_i,n_\ell\}$ (see I.4 in Table 3). Therefore the message in (A.9) must have been sent at time $t2 > t1$ and since ℓ sends messages only to nodes k for which $F_\ell(k)$ = UP it follows that $F_\ell(i)(t2+)$ = UP. But $F_\ell(i)$ could have gone to UP from READY only because of II.1.5, II.2.5, II.4.2, II.6.2, II.7.7, II.8.2 or II.9.2 in Table 3, and not because of I.3 and in all the above we have $n_\ell > z_\ell(i) = z_i(\ell)$. Since n_ℓ is nondecreasing and ℓ sends MSG(m,d) only with $m = n_\ell$, the assertion follows.

Lemma A.5

If

$$T\phi 2[t1,i,(\phi,m)] , \tag{A.10}$$

then $\forall t > t1+$, we have $\forall k$ s.t. $F_i(k)(t)$ = READY that $z_i(k)(t) \geq m$. Therefore, no link can be brought up by node i with number m after the node entered S2[m] (brought up means $F_i(k) \leftarrow$ UP).

Proof

If we have $F_i(k)(t1-) = \text{READY}$ and $z_i(k)(t1-) < m$, then at time $t1$, we have $F_i(k) \leftarrow \text{UP}$. If it is not, then $\forall t > t1$, we have $n_i(t) \geq m$ by Lemma A.2, so that only for nodes k with $z_i(k) \geq m$ it can happen that $F_i(k) \leftarrow \text{READY}$ after $t1$.

The next theorem completes the proof of Theorems 1, 2 and 3.

Theorem A.1

Let $PC(\bar{m})$, $PC(\bar{\bar{m}})$ be the instants of occurrence of two successive proper completions. Then

a) Theorem 3.

b) Consider any number $m1 \leq \bar{\bar{m}}$. Let $\tilde{m}$ be the highest number $\tilde{m} \leq m1$ such that $PC(\tilde{m})$ occurs. Let $LPC(\tilde{m},m1)$ be the time of occurrence of the last $PC(\tilde{m})$ such that $PC(\tilde{m}) \leq PC(\bar{m})$. If for any i,k, $t \leq PC(\bar{\bar{m}})$, we have either

$$N_i(k)(t) = m1 = \tilde{m},\ D_i(k)(t) \neq \infty,\quad s_i(t) \neq S3,\ n_i(t) = \tilde{m} \qquad \text{(A.11a)}$$

or

$$N_i(k)(t) = m1 > \tilde{m}\ , \qquad \text{(A.11b)}$$

then $\exists \tau1 \in [LPC(\tilde{m},m1),t)$ and $\tau2 \in (\tau_1,t)$ such that

$$[\tau1,k,SEND(m1,d1,i)] \qquad \text{(A,12a)}$$

$$[\tau2,i,MSG(m1,d2,k)] \qquad \text{(A.12b)}$$

with $d1 = D_i(k)(t) - d_{ik}(\tau2)$, $d2 = D_i(k)(t)$.

(Note: In words, the above insures that the message $(m1,d1)$ was sent and received no earlier than $LPC(\tilde{m},m1)$).

c) Consider any number $m1 \leq \bar{\bar{m}}$. Let $\tilde{m}$ be the highest number $\tilde{m} \leq m1$ such that $PC(\tilde{m})$ occurs. Let $LPC(\tilde{m},m1)$ be the time of occurrence of the last $PC(\tilde{m})$ such that $PC(\tilde{m}) \leq PC(m)$. Then

i) $[t1,i,MSG(m1,d1,\ell)]$, $[t2,i,MSG(m2,d2,\ell)]$ where $LPC(\tilde{m},m1) \leq t1 < t2 \leq PC(\bar{\bar{m}})$ and $d2 \neq \infty$ imply $m2 > m1$.

ii) If

$$T21[t1,i,(n1,n1)] \tag{A.13}$$

$$[t2,i,MSG(m,d,\ell)], \quad d \neq \infty \tag{A.14}$$

where $LPC(\tilde{m},n1) \leq t1 < t2 \leq PC(\bar{\bar{m}})$, then $m > n1$.

iii) A node i enters, between $LPC(\tilde{m},m)$ and $PC(\bar{\bar{m}})$, each of the following sets of states at most once

$$\{S1[m]\}, \{S2[m], S2[m]\}, \{S3[m]\} .$$

d) All "Facts" in Table 3 are correct.

e) i) The possible transitions at a node are the following, where $n2 \geq n1$ and $n3 > n1$: $T12[(n1,n2)]$, $T13[(n1,n2)]$, $T21[(n1,n1)]$, $T22[(n1,n3)]$, $T2\tilde{2}[(n1,n1)]$, $T23[(n1,n2)]$, $T\hat{2}3[(n1,n2)]$, $T32[(n1,n3)]$, $T\tilde{2}2[(n1,n3)]$.

ii) $T21[t,i,(n1,n1)]$, $p_k(t) = i$ implies $s_k(t) = S1[n1]$.

f) Theorem 1.

g) i) Suppose $T21[t,i,(n1,n1)]$ with $n1 = \bar{\bar{m}}$ and let $\tau1$ be the last time before t such that $T\phi2[\tau1,i,(\phi,n1)]$. Then we have $F_i(k)(\tau1) = UP$ if and only if $F_i(k)(\tau) = UP$, $\forall \tau \in [\tau1,t]$.

ii) If for some $t \in [PC(\bar{m}), PC(\bar{\bar{m}})]$ we have

$$T\phi2[t,i,(\phi,n2)], \quad n2 = \bar{\bar{m}} , \tag{A.15}$$

then

$$\exists \tau1 \in (t,PC(\bar{\bar{m}})) \text{ s.t. } T21[\tau1,i,(n2,n2)]$$

and

$$\nexists \tau2 \in (t,PC(\bar{\bar{m}})) \text{ s.t. } T23[\tau2,i] \text{ or } T2\tilde{2}[\tau2,i]. \tag{A.16}$$

h) If $\exists i,k,\ t \in (PC(\tilde{m}),PC(\bar{\bar{m}}))$ such that for some $\tau \in (PC(\tilde{m}),t]$ we have

$$[\tau,k,SEND(\bar{\bar{m}},d,i)],\quad d \neq \infty$$

and if i either has not received this message by time t or has $N_i(k)(t) = \bar{\bar{m}},\ D_i(k)(t) \neq \infty$, then $\exists t1 \in [t,PC(\bar{\bar{m}})]$ such that

$$s_i(t1) = S2[\bar{\bar{m}}] \ \underline{or}\ s_i(t1) = S3[\bar{\bar{m}}]\ . \tag{A.17}$$

Proof

As said before, the proof proceeds using a two-level induction. We first notice that a) holds at the time the network comes up for the first time. We call this PC(0). Then we assume that a) - h) hold at every time up and including $PC(\tilde{m})$. Next we prove that b) - h) hold until $PC(\bar{\bar{m}})$ and then show that a) holds at $PC(\bar{\bar{m}})$.

b) Observe that from Lemma A.2 b) and Property R8, by time $LPC(\tilde{m},m1)$ no node in the network has ever heard of a number $> \tilde{m}$. Therefore if (A.11b) holds, an appropriate message must have been sent and received after $LPC(\tilde{m},m1)$ and hence (A.12) holds.

On the other hand, observe that (A.11a) and Property R3 imply that $s_i(LPC(\tilde{m},m1)) \neq S3[\tilde{m}]$. Also note that the induction hypothesis assumes that a), namely Theorem 3, holds at time $LPC(\tilde{m},m1)$ and therefore at this time, first, no message $MSG(m,d)$ with $d \neq \infty$ is on its way to i and second, it cannot happen that $\{N_i(k) = \tilde{m}, D_i(k) \neq \infty\}$. But (A.11a) says that the latter occurs at time t and therefore, by step I.3 in Table 3, i must have received a message $MSG(\tilde{m},d)$ with $d \neq \infty$ after $LPC(\tilde{m})$ and hence (A.12b). Since no such message was on its way to i at $LPC(\tilde{m})$, A(12a) holds also.

c) Suppose c) i), ii) and iii) are true <u>for all nodes</u> in the network up to time $t-$. We prove c) i) and c) ii) for $t2 = t$ and then prove c) iii) for t.

i) If $d1 = \infty$, then $m2 > m1$ from Lemma A.3. It remains to prove the assertion for $d1 < \infty$. From Lemma A.2, we have $m2 \geq m1$. Suppose $d1 \neq \infty$ and $m2 = m1$. Then Lemmas A.3a) and A.2a) respectively, imply that $\nexists\, t3 \in (t1,t)$ such that $[i,t3,MSG(d3 = \infty,\ell)]$ or such that $[i,t3,MSG(m3,d3,\ell)]$, $m3 \neq m2 = m1$. Therefore the two messages received at $t1$ and $t2 = t$, can be taken as consecutive. So using b), $\exists\, t4 \in [LPC(\tilde{m},m1),t1)$, $t5 \in (t4,t)$ such that

$$Txy[t4,\ell,SEND(m1,d1,i)], \quad d1 \neq \infty \;, \tag{A.18}$$

$$T\alpha\beta[t5,\ell,SEND(m1,d2,i)], \quad d2 \neq \infty \;. \tag{A.19}$$

By R2, $Txy = T21$ or $T12$ or $T32$ or $T22$ or $T\tilde{2}2$ and same for $T\alpha\beta$. But by induction hypothesis on c) iii), node ℓ cannot enter the set of states $\{S2[m1], \tilde{S2}[m1]\}$ twice between $LPC(\tilde{m},m1)$ and t, so that the only possibilities are

$\{T12[t4,\ell]$ <u>OR</u> $T32[t4,\ell]$ <u>OR</u> $T22[t4,\ell]$ <u>OR</u> $T\tilde{2}2[t4,\ell]\}$ <u>AND</u> $\{T21[t5,\ell]\}$ and no other transition happens between $t4$ and $t5$. But in $T\phi2[t4,\ell]$, node ℓ sends a message to every neighbor except $p_\ell(t4+)$ and in $T21[t5,\ell]$ it sends a message only to $p_\ell(t5-)$ and since no other transition happens between $t4$ and $t5$ we have $p_\ell(t4+) = p_\ell(t5-)$. This contradicts (A.18), (A.19).

ii) If $F_i(\ell)(t1-)$ = DOWN or READY, then Lemma A.4 together with the facts that n_i is nondecreasing (by Lemma A.2b) and that $z_i(\ell)$ is established as in step I.4 of Table 3 show that the first message $MSG(m1,d1,\ell)$ that can be received by i from ℓ after $t1$ must have $m1 > n1$. Then the assertion follows from Lemma A.2a).

If $F_i(\ell)(t1-) = UP$, then step II.3.1 in Table 3 requires

$$N_i(\ell)(t1-) = n1 \tag{A.20}$$

and by the definition of $LPC(\tilde{m},n1)$ we have $n1 \geq \tilde{m}$.

If $D_i(\ell)(t1-) = \infty$, then $\exists t3 < t1$ (possibly $t3 < LPC(\tilde{m},n1)$) such that

$$[t3,i,MSG(n1,d1,\ell)], \quad d1 = \infty , \tag{A.21}$$

which together with (A.14) implies by Lemma A.3a) that $m > n1$.

If $D_i(\ell)(t1-) \neq \infty$, then from b) follows $\exists t3 \in [LPC(\tilde{m},n1),t1)$, such that

$$[t3,i,MSG(n1,d1,\ell)], \quad d1 < \infty \tag{A.22}$$

and the assertion follows from c) i).

iii) From Lemma A.2, n_i is nondecreasing, so that once n_i is increased, it cannot return to the old value.

From the algorithm, a node can leave $\{S2[m], \tilde{S2}[m]\}$ and not change $n_i = m$ only via T21 or T23 or $\tilde{T23}$. If T23 or $\tilde{T23}$, then R3 shows that it will strictly increase n_i when leaving S3[m]. If T21[(m,m)], then c) ii) shows that it cannot subsequently receive a message MSG(m,d) with $d \neq \infty$, and in order to enter S2[m], such a message must be received. Therefore, the statement holds for $\{\tilde{S2}[m], S2[m]\}$.

To S1[m] one enters only from S2[m], so that a node cannot enter S1[m] twice unless it enters $\{S2[m], \tilde{S2}[m]\}$ twice, so that the statement holds for S1[m].

If a node enters S3[m], by R3 it leaves it only with a higher n_i, so that it cannot come back with the same n_i.

d) The Fact in I.3 was proved in Lemma A.4. The Fact in I.4 follows from property 2.7.7 in Sec. 2.7. Next, observe from II.2.3, II.2.7, II.6.3 and II.9.3 in Table 3 that

$$T\phi3[i,(d1,d2),(p1,p2)] \tag{A.23}$$

implies $d2 = \infty$, $p2 = nil$, so Fact 32 is correct. Facts 13, 12, 23 and $\tilde{2}3$ follow from Lemma A.2a) and A.2c), since if MSG is received at i at time t and T13 or T12 or T23 or $T\tilde{2}3$ happen, then

$$m \triangleq \text{number received by } i \text{ at } t \text{ on } p_i(t-) \geq M_i(t-,p_i(t-)). \tag{A.24}$$

Fact 21 is correct, since if $T\phi2[i,(d1,d2)]$, then $d2 < \infty$ and since $p_i = nil$ iff $s_i = S3$.

e) i) The assertion follows immediately from Lemma A.2 b) and from checking changes on n_i in Table 3.

ii) Recall that we are always considering times until $PC(\bar{\bar{m}})$. Observe from II.3.1 in Table 3 that

$$T21[t,i,(n1,n1)] \tag{A.25}$$

implies that $N_i(\ell)(t-) = n1$ for all ℓ with $F_i(\ell) = UP$, and since from II.3.7 in Table 3 $p_k(t) = i$ implies $F_i(k) = UP$, we have $N_i(k)(t-) = n1$. Note further that $D_i(k)(t-) \neq \infty$, since otherwise k was some time before t in $S3[n1]$ and could set $p_k \leftarrow i$ only if i sent to k a message MSG with number strictly higher than $n1$. But $N_i(k)(t-) = n1$, $D_i(k)(t-) \neq \infty$ implies from b) that $\exists \tau \in [LPC(\bar{\bar{m}},n1),t)$ such that

$$Txy[\tau,k,SEND(n1,d,i)], \quad d \neq \infty \; . \tag{A.26}$$

Now if $p_k(\tau-) \neq i$, then $Txy = T12$, but in order for $p_k(t) = i \neq p_k(\tau-)$, k must have performed $T21[\tau1,k]$ at some

$\tau 1 \in (\tau,t)$. On the other hand, if $p_k(\tau-) = i$, then $Txy = T21$. Therefore k performed

$$T21[\eta,k,(n1,n1),(p1,p2)], \quad p2 = i \tag{A.27}$$

at some time $\eta \in [LPC(\tilde{m},n1),t)$. So $s_k(\eta+) = S1[n1]$.

From e) i), the fact that until t node k receives no number higher than $n1$ and $p_k(t) = i$, one can easily see that k remains in $S1[n1]$ until time t.

f) We refer to the properties to be proven here as tree properties. If $p_i = k$, we say that i is a <u>predecessor</u> of k and k the <u>successor</u> of i. Also, we look at the concatenation (n_i,s_i) and write $(n_i,s_i) \geq (n_k,s_k)$ if $n_i \geq n_k$ <u>and</u> if $n_i = n_k$ implies $s_i \geq s_k$. Using this notation observe from e) i), that

$$Txy[i,(n1,n2)]$$

implies $(n2,y) \geq (n1,x)$ except when $Txy = T21$.

As before, we prove the tree properties by induction, assuming that they hold up to time $t-$ and showing that any possible change at time t preserves the properties. The changes of interest here are in the quantities n_i, s_i, p_i, d_i.

Let us consider all possible transitions:

$T2\tilde{2}[t,i]$; only s_i changes, $s_i(t+) = s_i(t-)$, so "trees" properties are preserved.

$T13[t,i]$, $T23[t,i]$, $T\tilde{2}3[t,i]$; then $p_i(t+) = nil$, so no successor at $t+$. Also by Lemma A.2 and induction hypothesis follows that if $p_k(t) = i$, then

$$(n_i,s_i)(t+) > (n_i,s_i)(t-) \geq (n_k,s_k)(t) , \tag{A.28}$$

so properties are preserved for all predecessors.

$T12[t,i]$, $T22[t,i]$, $\tilde{T}22[t,i]$ (change d_i, s_i and possibly n_i; no change in p_i). Regarding predecessors, the proof evolves as for T13. Regarding p_i, we see that

$$Txy[t,i,MSG(m,d,\ell),(n1,n2),(p1,p1)] \, , \tag{A.29}$$

where $Txy = T12$ or $T22$ or $\tilde{T}22$, implies from steps II.1.1, II.4.1, II.8.1 in Table 3 that $\ell = p1$, $d \neq \infty$ and from steps II.1.4, II.4.2, II.8.2 that $m = n2$. From b) and R2, this implies that $\exists \tau \in [LPC(\tilde{m},m),t)$ such that $s_{p1}(\tau) = S2[n2]$. Now, if on (τ,t), p1 stayed in $S2[n2]$ or performed any transition except $T21[p1,(n2,n2)]$, then $T12[i]$ or $T22[i]$ or $\tilde{T}22[i]$ preserve the tree properties. We want to show by contradiction that p_i could not have performed T21 on (τ,t). Suppose

$$T21[\tau 1,p1,(n2,n2)], \quad \tau < \tau 1 < t \, , \tag{A.30}$$

then by step II.3.1 of Table 3 we have $N_{p1}(i)(\tau 1) = n2$. Now we distinguish between two cases:

If $D_{p1}(\tau 1) \neq \infty$, then by b), $\exists \tau 2 \in (LPC(\tilde{m},n2),\tau 1)$ such that

$$[\tau 2,i,SEND(n2,d,p1)] \, , \quad d \neq \infty \tag{A.31}$$

which by R2 implies that $s_i(\tau 2-) = S2[n2]$ or $s_i(\tau 2+) = S2[n2]$. But $T12[t,i,(n1,n2)]$ or $T22[t,i,(n1,n2)]$ or $\tilde{T}22[t,i,(n1,n2)]$ says that i enters $S2[n2]$ at time t which contradicts c) iii).

If $D_{p1}(i)(\tau 1) = \infty$, then for some $\tau 2 < \tau 1$ (not necessarily $\tau 2 > LPC(\tilde{m},n2)$)

$$[\tau 2,i,SEND(n2,d,p1)], \quad d = \infty$$

which implies that $s_i(\tau 2+) = S3[n2]$. But $s_i(t+) = S2[n2]$ and $\tau 2 < t$, which is impossible by R3 and Lemma A.2.

T32[t,i,(n1,n2),(nil,p1)]. Regarding predecessors the tree properties are preserved since n2 > n1. Regarding successor, the above implies that $\exists \tau \in (LPC(\tilde{m},n2),t)$

$$[\tau, p1, SEND(n2,d,i)] .$$

Now, from Lemma A.2, $n_{p1}(t) \geq n_{p1}(\tau)$. From R2, $n_{p1}(\tau) = n2$. Now, if $n_{p1}(t) > n2$, then

$$(n_{p1}, s_{p1})(t) > (n_i, s_i)(t+) .$$

If on the other hand $n_{p1}(t) = n2$, then the same argument as for T12, T22 shows that p1 was in S2[n] sometime before t and could not return to S1[n2] in the meantime, so that

$$(n_{p1}, s_{p1})(t) \geq (n_i, s_i)(t+) .$$

In addition to the above, since here there is a change in p_i from nil to $\neq$nil, we have to check that this change does not close a loop. This is seen from the fact that every node k upstream from i at time t has

$$(n_k, s_k)(t) \leq (n_i, s_i)(t-) = (n1,3) < (n2,2) = (n_i, s_i)(t+)$$

and every node ℓ downstream from p1 has

$$(n_\ell, s_\ell)(t) \geq (n_{p1}, s_{p1})(t) \geq (n2,2).$$

T21[t,i,(n1,n1),(p1,p2),(d1,d1)]. If $p_k(t) = i$, then from e) ii) follows that $s_k(t) = S1[n1]$, so

$$(n_i, s_i)(t+) = (n_k, s_k)(t)$$

Regarding successor, steps II.3.1 and II.3.7 of Table 3 show that $N_i(p2)(t-) = n1$, $D_i(p2)(t-) \neq \infty$, so that from b), $\exists \tau \in [LPC(\tilde{m},m),t)$ such that

$$[\tau, p2, \mathrm{SEND}(m,d,i)]$$

with $m = n1 = n_{p2}(\tau+)$, $d = d_{p2}(\tau+) = D_i(p2)(t-) - d_{i,p2}(\tau)$. Therefore from Lemma A.2,

$$(n_{p2}, s_{p2})(t) \geq (n1,1) = (n_i, s_i)(t+) \; .$$

Now suppose that the change in p_i closes a loop at $t+$. Then the last expression and the induction hypothesis show that at time $t+$

$$(n_{p_\ell}, s_{p_\ell}) \geq (n_\ell, s_\ell)$$

for all nodes ℓ around the loop, so that (n,s) must be constant around the loop, namely

$$(n,s) = (n1,1)$$

around the loop. Therefore $s_{p2}(t) = S1[n1]$. But by R2, $s_{p2}(\tau-) = s_{p2}(\tau+) = S2[n1]$ where τ is defined above, so by c) iii), node p2 could not enter again $S2[n1]$ between $\tau+$ and t, so

$$d_{p2}(t) = d_{p2}(\tau+) = D_i(p2)(t-) - d_{i,p2}(\tau) .$$

But from steps II.3.2 and II.3.7 of Table 3

$$d1 \geq D_i(p2)(t-) = d_{p2}(t) + d_{i,p2}(\tau)$$

which from Assumption 2.7.2 implies that

$$d1 = d_i(t+) > d_{p2}(t) \; .$$

On the other hand, the induction hypothesis implies that since $(n_\ell, s_\ell) \equiv (n1,1)$ around the loop, we have

$$d_\ell(t) \geq d_{p_\ell}(t)$$

for all $\ell \neq i$ around the loop and this provides a contradiction, therefore no loop is closed by the change in p_i.

g) i) During $(\tau 1,t)$, no link is brought up by i because of Lemma A.4. If there are failures, let $\tau 3$ be the first time on $(\tau 1,t)$ such that

$$[\tau 3,i,\mathrm{FAIL}(k)] .$$

Then $T23[\tau 3,i,(n1,n1)]$ or $T2\tilde{2}[\tau 3,i,(n1,n1)]$ happen with $n1 = \bar{\bar{m}}$. In either case, e) i) shows that to exit $S3[n1]$ or $S\tilde{2}[n1]$, one has to increase n_i, so that it is not possible that

$$T21[t,i,(n1,n1)] .$$

So no failure can occur.

ii) Consider the sequences of nodes and instants

$$i = i_o,i_1,i_2,\ldots,i_s = \mathrm{SINK}$$

$$t = t_o > t_1 > t_2 > \ldots > t_s$$

such that

$$T\phi 2[t_u,i_u,(\phi,n2),(p1_u,p2_u)]$$

where $n2 = \bar{\bar{m}}$ and $p2_u = i_{u+1}$. There must have existed such sequences if $T\phi 2[i_o]$. Suppose $\nexists \tau \in [t_o,PC(\bar{\bar{m}})]$ such that

$$T21[\tau,i_o,(n2,n2)] .$$

We want to show that $\nexists \tau 1 \in [t_1,PC(\bar{\bar{m}})]$ such that

$$T21[\tau 1,i_1,(n2,n2)] .$$

If there existed such a $\tau 1$, it follows from g) i) that $F_{i_1}(i_o)(\tau 1) = UP$.

We want to show now that $\nexists \tau 2 < \tau 1$ such that

$$[\tau 2,i_o,\mathrm{SEND}(n2,d,i_1)],\quad d = \infty ,$$

and $\nexists \tau 3 \in (PC(\bar{\bar{m}}),\tau 1)$ such that such a message with $d \neq \infty$ is sent. For $\tau 2 < t_o$, this follows respectively from R2, R3 and

R2, c) iii). For $\tau 2 = t_o$, it follows from the fact that

$$p_{i_o}(t_o+) = i_1 .$$

For $\tau 2 \in (t_o, PC(\bar{\bar{m}}))$, the only possibilities for i_o if T21 does not happen, are to stay in $S2[\bar{\bar{m}}]$ or $T\tilde{22}[(n2,n2)]$, or $T23[(n2,n2)]$, or $T\tilde{23}[(n2,n2)]$. In all cases i_o will not send any message to i_1.

The above show that $N_{i_1}(i_o)(\tau 1-) \neq \bar{\bar{m}} = n2$ so that

$$T21[\tau 1, i_1, (n2,n2)]$$

is impossible. Repeating the proof, it follows that $\nexists \tau_s$ such that

$$T21[\tau_s, SINK, (n2,n2)], \quad n2 = \bar{\bar{m}} ,$$

which contradicts the assumption that there is a proper completion at time $PC(\bar{\bar{m}})$. This proves the first part of g) ii). The second part follows because $T21[\tau 1, i, (n2,n2)]$, $n = \bar{\bar{m}}$ is not possible if $T23[i,(n2,n2)]$ or $T\tilde{22}[i,(n2,n2)]$ happen.

h) If $[\tau, k, SEND(\bar{\bar{m}}, d \neq \infty, i)]$, then $F_k(i)(\tau) = UP$ and by R2 either

$$Tx2[\tau, k, (\phi, n2)], \quad n2 = \bar{\bar{m}}, \quad x = 1,2,3$$

or

$$T21[\tau 1, k, (n2,n2)], \quad n2 = \bar{\bar{m}} .$$

If Tx2 then g) ii) implies $\exists \tau 2 \in (\tau, PC(\bar{\bar{m}}))$ such that

$$T21[\tau 2, k, (n2,n2)], \quad n2 = \bar{\bar{m}}$$

and $F_k(i)(\tau 1) = UP$. Therefore T21 happens at node k at some time ($\tau 1$ or $\tau 2$). Call this time η. We have then $N_k(i)(\eta) = \bar{\bar{m}}$. By b) either $\exists \tau 3 \in [PC(\bar{\bar{m}}), \eta)$ such that

$$[\tau 3, i, \mathrm{SEND}(\bar{\bar{m}}, d \neq \infty, k)]$$

or $\exists \tau 4 < \eta$ such that

$$[\tau 4, i, \mathrm{SEND}(\bar{\bar{m}}, d = \infty, k)].$$

But by R2, this means that i is at some time before η in $S3[\bar{\bar{m}}]$ or is at some time between $PC(\bar{m})$ and $PC(\bar{\bar{m}})$ in $S2[\bar{\bar{m}}]$. If the first holds, node i will stay in $S3[\bar{\bar{m}}]$ at least until $PC(\bar{\bar{m}})$. If the latter holds, then by g) ii) it must perform $T21[i,(n2,n2)]$ before $PC(\bar{\bar{m}})$. But since it still has $N_i(k)(t) = \bar{\bar{m}}$, $D_i(k)(t) \neq \infty$ or has not received yet the message by time t, property c) i) implies that node i could not perform $T21[i,(n2,n2)]$ before time t. Therefore it will perform later, so q.e.d.

<u>Proof that a) holds at time $PC(\bar{\bar{m}})$</u>

i) Node i cannot be in $S2[\bar{\bar{m}}]$ because of g) ii) and c) iii). It cannot be in $\tilde{S2}[\bar{\bar{m}}]$ because it must have been in $S2[\bar{\bar{m}}]$ before and because of g) ii).

ii) Take $t = PC(\bar{\bar{m}})$ in h). Then h) says that

$$s_i(PC(\bar{\bar{m}})) = S2[\bar{\bar{m}}] \text{ or } S3[\bar{\bar{m}}].$$

But g) ii) and c) iii) imply that $s_i(PC(\bar{\bar{m}})) \neq S2[\bar{\bar{m}}]$.

iii) Follows by contradiction, because if we had

$$N_i(k)(PC(\bar{\bar{m}})) = \bar{\bar{m}}, \quad D_i(k)(PC(\bar{\bar{m}})) \neq \infty ,$$

it follows by taking $t = PC(\bar{\bar{m}})$ in h) that

$$s_i(PC(\bar{\bar{m}})) = S2[\bar{\bar{m}}] \text{ or } S3[\bar{\bar{m}}] .$$

This completes the proof of Theorem A.1.

Appendix B

In Appendix A we have proved Theorems 1, 2 and 3. This appendix is devoted to proofs of the remaining statements, namely Theorem 4 (normal activity) and Propositions 1 and 2 that lead to the recovery theorem, Theorem 5. The proofs are organized as follows: Lemma B.0 is preliminary and shows that on any link (i,ℓ) the only two "stable" situations are $\{F_i(\ell) = F_\ell(i) = \text{DOWN}\}$ or $\{F_i(\ell) \neq \text{DOWN},\ F_\ell(i) \neq \text{DOWN}\}$. Lemmas B.1 and B.2 prove Proposition 1, Lemma B.3 proves Theorem 4, and the Proposition 2 is proved by the series of four lemmas B.4 to B.7.

Lemma B.0

If $F_i(\ell)(t1) = \text{DOWN}$, $F_\ell(i)(t1) \neq \text{DOWN}$, then in finite time after t1 we have either $F_i(\ell) = F_\ell(i) = \text{DOWN}$ or $\{F_i(\ell) \neq \text{DOWN}$ and $F_\ell(i) \neq \text{DOWN}\}$.

Proof

If $F_\ell(i)(t1) = \text{READY}$, then i and ℓ arrived to this situation from $\{F_\ell(i) = F_i(\ell) = \text{DOWN}\}$ or $\{F_\ell(i) = F_i(\ell) = \text{READY}\}$ or $\{F_\ell(i) = \text{READY},\ F_i(\ell) = \text{UP}\}$. Then assumptions 2.7.9 imply the assertion.

If $F_\ell(i)(t1) = \text{UP}$, then i and ℓ arrived to this situation from $\{F_\ell(i) = \text{READY},\ F_i(\ell) = \text{DOWN}\}$ or $\{F_\ell(i) = F_i(\ell) = \text{UP}\}$, or $\{F_\ell(i) = \text{UP},\ F_i(\ell) = \text{READY}\}$. In the first case, the discussion reduces to the first part of the proof, whereas for the second and third case, assertion 2.7.9 a) in Sec. 2.7 proves the assertion.

Lemma B.1

Proposition 1(a).

Proof

Clearly, $n_i(t1-) < m2$ for all i. Therefore (10) may happen only at or after t1.

Let

$$A(t) = \{i: i \in L(t) \text{ and } i \text{ effected (10) with } t2_i < t\} .$$

If $\exists t2$ such that $A(t2) = L(t2)$, then the proof is complete. Otherwise, for a given $t3$, we will show (by contradiction) that $\exists t$, $t3 < t < \infty$ such that

$$A(t) \supset A(t3) \text{ and } A(t) \neq A(t3) . \qquad \text{(B.1)}$$

Hence by induction, the set $A(t)$ keeps growing until it equals $L(t)$.

Since there are no pertinent topological changes and all $i \in A(t)$ have $n_i(t) = m2$, property R10 implies that the set $A(t)$ is nondecreasing as t increases. Therefore to prove part i) of Proposition 1(a) it is sufficient to show that the following cannot hold:

$$\forall t > t3, \quad A(t) = A(t3) \neq L(t) \qquad \text{(B.2)}$$

Let

$B(t) = \{i | i \in L(t) \text{ and } i \notin A(t)\}$,

$A'(t) = \{i | i \in A(t)$ and i has a potentially working link to a node of $B(t)\}$,

$B'(t) = \{i | i \in B(t)$ and i has a potentially working link to a node of $A(t)\}$.

The following three claims will contradict (B.2).

Claim 1

If (B.2) holds, then $\exists t4 \in (t3,\infty)$ such that $\forall j \in B'(t4), \exists t4_j < t4$ such that $[t4_j, j, MSG(m2)]$, (i.e. all nodes of $B'(t4)$ receive m2 in finite time).

Proof of Claim 1

At time $t2_i < t3$, node $i \in A'(t2_i)$ performs transition (10). Now observe that since no pertinent topological changes occur, property R10 implies that for all ℓ, $F_i(\ell)$ cannot be changed from or to DOWN after $t2_i$. Therefore if $F_i(\ell)(t2_i-) = \text{DOWN}$ then $F_i(\ell)(t) = \text{DOWN}$ for $t \geq t2_i$ and if $F_i(\ell)(t2_i-)$ = READY or UP, then $F_i(\ell)(t)$ = UP for $t > t2_i$ (see II.1.5, II.4.2, II.7.7, II.8.2 in Table 3). For links (i,ℓ), where $i \in A'(t2_i)$, $\ell \in B'(t2_i)$ and $F_i(\ell)(t2_i+)$ = UP, observe from II.1.6 in Table 3 that if $p_i(t2_i) \neq \ell$, then

$$[t2_i, i, \mathrm{SEND}(m2,\ell)] .$$

Since by Lemma A.2c) we have

$$p_i(t2_i) \neq B(t2_i)$$

and since property 2.7.9 Sec. 2.7 insures that the above message will arrive, there is a time t4 for which all nodes j that were in $B'(t2_i)$ for some i, either are not in $B'(t4)$ anymore or have received MSG(m2). Also observe that $B'(t4)$ cannot be empty, since then B.2 is contradicted.

Let $t5_{jk}$ denote the time at which $j \in B'(t4)$ receives MSG(m2,k), where $k \in A'(t4)$. If $\exists j \in B'(t4)$ such that $p_j(t5_{jk}) = k$ for some $k \in A'(t4)$ then from II.1.1, II.4.1, II.8.1 in Table 3, the transition $T\phi2[j,(\phi,m2)]$ occurs, contradicting (B.2), q.e.d. Otherwise,

Claim 2

If $j \in B'(t4)$ such that $p_j(t5_{jk}) \neq k$ then $\forall t > t5_{jk}$, $p_j(t) \neq k$.

Proof of Claim 2

Suppose

$$Txy[t,j,(p1,p2 = k)], \quad t > t5_{jk} .$$

If $x \neq 3$, by R5 Txy = T13 or T21 or T23 or $\tilde{T}23$,

But $\tilde{T}23$, T13, T23 $\Rightarrow$ p2 = nil $\neq$ k, therefore this cannot happen.

T21 $\Rightarrow \forall q$, $N_j(q)(t) = n_j < m2$, but $N_j(k)(t) = m2$, hence T21 cannot happen.

If x = 3 then T32[t, j, MSG(m2)] happens, contradicting (B.2), q.e.d. Claim 2.

Claim 3

In finite time, all nodes $i \in B(t4)$ will effect $T\phi3[i,(\phi,m)]$, $m \leq m1$ without effecting $T3\phi$ thereafer.

Proof of Claim 3

n_i is updated in T12, T13, T22, T23 and T32 only. For all $i \in B(t4)$, $T\phi2[i,(\phi,m2)]$ does not occur because of (B.2), and $T\phi3[i,(\phi, m2)]$ does not occur because there are no pertinent topological changes. Hence,

$$\forall i \in B(t4) \text{ and } \forall t > t4, \; n_i(t) \leq m1 .$$

Since after t4 no update cycles with $m \leq m1$ are started by Theorem 2(ii), the number of messages with $d < \infty$ generated by the nodes of $B(t4)$ is finite. Similarly, since the number of arcs is finite, the number of messages FAIL is also finite. Consider $B(t4)$ after all these messages are generated and received. Then $\forall i \in B(t4)$, $T3\phi[i]$ cannot occur and $Txy[i,(p1,p2 \neq p1)]$ implies $p2 = nil$. Then

1. if $\forall k \in B(t4)$, $p_k = nil$, then q.e.d. Claim 3;
2. otherwise, after a sufficiently long period of time t_{mx}, by Claim 2 and Theorem A1, there exist k and i such that:

   $$i,k \in B(t3), \; p_k(t_{mx}) = i \text{ and } p_i(t_{mx}) = nil .$$

   When p_i was set to nil, $Txy[i,SEND(m,d = \infty,k)]$ occurs. At t_{mx} this message is not yet received by k because $p_k(t_{mx}) = i$. After this message is received, node k effects $T\phi3$, enters S3 and does not leave it anymore. By induction, q.e.d. Claim 3.

The proof of Proposition 1(a)(i) is completed as follows. Consider a node $j \in B'(t4)$. Define $t3_j$ to be the time at which $T\phi3[t3_j,j]$ occurs by Claim 3. But if $t3_j < t5_{jk}$ then $T3\bar{2}[t5_{jk}, j]$ happens,

if $t3_j > t5_{jk}$ then $T32[t3_j, j]$ occurs, and $t3_j \neq t5_{jk}$,

which contradicts (B.2), q.e.d.

To prove part (ii) of Proposition 1(a), we investigate further the situation in $L(t2)$ at time $t2$. Observe that since all nodes in $L(t2)$ have $n_i = m2$, and no pertinent topological changes happen, it follows from R10 and Lemma B.0 that for any link (i,ℓ) such that $i \in L(t2)$, $\ell \in L(t2)$, it <u>cannot</u> happen that at time $t2$ we have $F_i(\ell) = \text{DOWN}$, $F_\ell(i) \neq \text{DOWN}$. Also $F_i(\ell) = \text{READY}$ is not possible, because lack of pertinent topological changes imply that $F_i(\ell)(t2_i-) = \text{READY}$ as well, and then II.1.5 in Table 3 shows that, for example $F_i(\ell)(t2_i+) = \text{UP}$ and therefore $F_i(\ell)(t2) = \text{UP}$. Therefore, for links (i,ℓ) connecting nodes in $L(t2)$, the only possibilities <u>at time $t2$</u> are $\{F_i(\ell) = F_\ell(i) = \text{DOWN}\}$, $\{F_i(\ell) = F_\ell(i) = \text{UP}\}$, hence Proposition 1(a)(ii) is proved.

Next, assuming Proposition 1(a) which was proved by Lemma B.1, we now prove Proposition 1(b).

<u>Lemma B.2</u>

Let $L(t)$ be as in Lemma B.1, and suppose that a new cycle $T\phi2[t1, \text{SINK}, (\phi, m1)]$ is started. Suppose also that no pertinent topological changes have happened before $t1$ while $n_{\text{SINK}} = m1$ and that no such changes will take place after $t1$ for a sufficiently long period of time. Define $t2_i$ to be the smallest time t such that

$$T\phi2[t, i, (\phi, m1)], \quad t > t1$$

occurs. Suppose also there exists $t2$, $t1 < t2 < \infty$ such that for all $i \in L(t2)$

$$T\phi2[t2_i, i, (\phi, m1)]$$

occurs with $t1 \leq t2_i \leq t2$, and $t2 = \max_{t2_i < \infty} (t2_i)$.

i) There exists a time $t3 < \infty$ such that $t2 < t3$ and that

$T21[t3,SINK,(m1,m1)]$ occurs;

ii) $\forall t \in [t2,t3]$, we have $H(t) = L(t) = H(t2)$;

iii) $RG(t3)$ for the nodes in $L(t3)$ is a single tree rooted at SINK.

Proof

We prove first that there is $PC(m1)$ after $t1$, then we show that there is no $PC(m1)$ between $t1$ and $t2$.

Since there are no pertinent topological changes, after entering $S2[m1]$ at $t2_i$ each node $i \in L(t2)$ can only perform transitions between states S1 and S2. Furthermore, by Theorem 1(i), after $t2$, these nodes form a single tree rooted at SINK. Consider a time t', $t' > t2$. Since there are no pertinent topological changes, $L(t') = L(t2)$. Also, by Theorem 2(iii), if a node $i \in L(t2)$ enters $S2[m1]$ after $t2$, $PC(m1)$ has occurred after $t1$.

1. If $\forall i \in L(t')$, $s_i(t') = S1$ then there exists $t3$, $t1 < t3 < t'$ such that $T21[t3,SINK,(m1,m1)]$ occurred;

2. otherwise, consider a node k such that $s_k(t') = S2$,

$$\forall j \text{ if } p_j(t') = k, \text{ then } s_j(t') = S1 \tag{B.3}$$

such a node k always exists. Classify the neighbors of k into:

$$A = \{i: F_i(k)(t') = UP \text{ and } s_i(t') = S1\}$$
$$B = \{i: F_i(k)(t') = UP \text{ and } s_i(t') = S2\}.$$

At some time in the interval $[t1,t']$, the nodes in A have sent messages $MSG(m1,d \neq \infty)$ to all their neighbors. At some time in the same interval, those in B have sent such messages to all their neighbors except $p_j(t')$. Hence by (B.3), k will receive messages $MSG(m1,d \neq \infty)$ from all its neighbors, at a finite time, say $t4$. Then

2.1 if $s_k(t4+) = S2$ means that $\exists i$ with $F_k(i)(t4) = UP$ such that $N_k(i)(t4) =$ which implies that $T21[k,(m1,m1)]$ occurred in the interval $[t1,t4]$, hence by Theorem 2(iii), PC(m1) occurred between t1 and t4;

2.2 if $s_k(t4+) = S1$, by induction, PC(m1) will occur in finite time after t1.

We show next that PC(m1) cannot happen in [t1,t2]. Suppose that at t5, the first PC(m1) after t1 occurs. Let k be a node such that $t2_k < t5$ and $k \in L(t2)$, hence since there are no pertinent failures, there exists a $j \in L(t2)$ such that $F_j(k)(t2_j) = F_j(k)(t5) = UP$. But j sent to k a message $MSG(m1,d \neq \infty)$ in the interval $[t2_j,t5]$; on the other hand by Theorem 3 such a node k does not exist.

Since there are no pertinent topological changes, we have $L(t2) = L(t3)$, and according to Theorem 1(i) these nodes have preferred links forming a single tree rooted at SINK and hence iii).

Finally, looking at the situation in the network at time t2 as described in Lemma B.1, and for all $t \in [t2,t3]$, we observe that for all (i,ℓ) for which $F_i(\ell)(t2) = UP$ we must have $F_i(\ell)(t) = UP$ and if $F_i(\ell)(t2) = DOWN$, we must have $F_i(\ell)(t) = DOWN$. This completes the proof of ii).

<u>Lemma B.3</u>

Theorem 4.

<u>Proof</u>

By the Algorithm, a new cycle $T12[t1,SINK,(m1,m1)]$ can start only if all previous cycles with the same counter number m1 were properly completed. Since cycle counter numbers are non-decreasing, the first cycle with m1 was started at a time, say t', by

$$T12[t',SINK,(m0,m1)], \quad m1 > m0 .$$

This transition satisfies the condition of Proposition 1. Hence in a finite time, say t", the cycle is properly completed, L(t") forms a tree rooted at SINK, all $i \in L(t")$ have $n_i = ml$, and since there are no pertinent topological changes, for all $t \geq t"$:

1. $H(t) = L(t) = L(t")$ q.e.d. Theorem 4(b), and
2. by Theorem 1(i) all nodes $i \in L(t)$ form a single tree rooted at SINK, q.e.d. Theorem 4(d,ii).

Define A_k to be the set of nodes that are on the tree at time t1, at a distance of k nodes from the SINK. A_o = SINK and it is assumed by Theorem 4 that T12[$t2_{SINK}$ = t1,SINK,(ml,ml)] occurs. Suppose all i A_k effect T12[$t2_i$,i,(ml,ml)], sending messages MSG(ml) to all $j \in A_{k+1}$ through their p_j(t1). But since there are no pertinent topological changes after t1, p_j can only change by T21, and since s_j(t1) = S1, only after T12. Then, all $j \in A_{k+1}$ will receive messages MSG(ml) at a finite time $t2_j$ from $p_j(t2_j)$, which trigger the occurrence of T12[$t2_j$,j,(ml,ml) and by induction on k, q.e.d. Theorem 4c).

Theorem 4(d,i) follows directly from Lemma B.2 by assuming Theorem 4(c). Theorem 4(a) follows directly from the algorithm for SINK. This completes the proof.

Proposition 2 will be proved by Lemmas B.4 and B.7. When an REQ(ml) is generated, it is placed in the queue for processing. If, when the REQ(ml) is processed, the node is at S2, $\tilde{S2}$ or S1, then an REQ(ml) is sent by this node to its current preferred link. The proof of Proposition 2 for these cases is given in Lemma B.5 (for S2 or $\tilde{S2}$) and Lemma B.7 (for S1). Lemma B.6 proves the proposition for the case where there is a node in state S3[ml]. Lemma B.4 is used to simplify proofs.

Lemma B.4

If a REQ(ml) is generated, then either:

1. REQ(ml) is processed only by nodes having $n_i = ml$, and all nodes j have $n_j \leq ml$, or

2. a REQ(ml) arrived at SINK.

Proof

By Theorem 1(ii) and by the Algorithm, REQ(ml) is not received (i.e. processed) by a node i with $n_i < ml$. On the other hand, if there exists a node i with $n_i > ml$, the SINK started a cycle with $m > ml$; this can happen only following the arrival of REQ(ml) to SINK, q.e.d.

Lemma B.5

If a node i sends REQ(ml) while $s_i = S2[ml]$ or $\tilde{S2}[ml]$, then a REQ(ml) arrived or will arrive at SINK in finite time.

Proof

Consider the strings of nodes and instants

$$i = i_o, i_1, i_2, \ldots, i_m = \text{SINK}$$

$$t_o > t_1 > t_2 > \ldots > t_m$$

such that

$$T\phi 2[t_u, i_u, (\phi, n2), (p1_u, p2_u)] ,$$

where $n2 = 1$, $p2_u = i_{u+1}$. There must exist such a string if $s_i = S2[ml]$ or $\tilde{S2}[ml]$. The string has no loops, otherwise Lemma B.4, Theorems 1, 2 or 4 will be contradicted.

Suppose that at time $t2_u$, a node i_u sends REQ(ml) to i_{u+1}. Suppose also that in the interval $[t_u, t2_u]$, node i_u effects no transition except possibly $\tilde{T22}$. After t_{u+1}, the first transition executed by i_{u+1} could be

$T22[i_{u+1}]$; q.e.d. by Theorem 3 and Lemma B.4.

$T\tilde{22}[i_{u+1}]$, in which case a failure is detected by i_{u+1} and REQ(ml) sent to i_{u+2}.

$T21[i_{u+1}]$; this transition is executed only after receiving a message from i_u. Such a message is sent by i_u when $T21[i_u]$ happens, i.e. after i_u has sent REQ(ml). Since FIFO is preserved, i_{u+1} will receive and therefore send REQ(ml) to i_{u+2} before $T21[i_{u+1}]$ happens, i.e. while $s_{i_{u+1}} = S2$.

$T23[i_{u+1}]$; in this case there exist i_r, $r > i+1$ such that $T\tilde{22}[i_r]$ and i_r sends REQ(ml) to i_{r+1}.

Thus by induction, REQ(ml) arrived or will arrive at SINK in finite time.

<u>Lemma B.6</u>

If there exists a node that effects $T\phi3[(\phi,ml)]$, then a REQ(ml) arrived or will arrive at SINK in finite time.

<u>Proof</u>

Let PC_j, $(j = 0,1,2,...)$ denote the j-th occurrence of PC[ml]. Given a node i and a time t such that $T\phi2[i,(\phi,ml)]$ has occurred before t, if PC_j is the last PC[ml] before t after which $T\phi2[i,(\phi,ml)]$ occurred, then define $E_i(t) = j+1$.

By Lemma B.4, we have to prove only the case in which $n_i \leq ml$ for all i. Thus, if a node i is in state S3[ml], this node will not execute any further transitions.

Property

Given a time t, suppose $p_i(t) = k$ and $n_i(t) = n_k(t) = m1$, then

$$E_i(t) \le E_k(t) .$$

This can be proved as follows:

Suppose that prior to t and after PC_a, p_i was last set to be k. This can be done only by T21[i] or T32[i]. Since at PC_a, $s_i \ne S2[m1]$ (by Theorem 3) this implies that Tϕ2[i] occurred after PC_a and Tϕ2[i] cannot occur again before t because this will set again p_i. Hence $E_i(t) = a+1$. The occurrence of T21[i] or T32[i] implies that a message from k with $d < \infty$ arrived at i after PC_a. By Theorem 3, this message was sent after PC_a, this being possible only if k effected Tϕ2[k] after PC_a. Since L_k is non-decreasing then $E_k(t) \ge a+1$.

Since after a node effected Tϕ3[(ϕ,m1)] the same node cannot perform any further transitions, only a finite number of transitions Tϕ3[(ϕ,m1)] can be executed in the network. If Tϕ3[(ϕ,m1)] happens, there exists a node which detects a failure in its best link and executes Tϕ3[(m1,m1)]. Define B1 as the set of nodes for which Tϕ3[(m1,m1)] happens, this is

$$B1 = \{i: T\phi 3[t_i, i, (m1,m1)] \text{ happens}\} .$$

Define B2 as the subset of B1 for which Tϕ3[(m1,m1)] happens with the highest E_i, i.e.

$$B2 = \{j: j \in B1 \text{ and } (E_j(t_j) = \max_{i \in A} E_i(t_i))\} ,$$

<u>Case 1</u>: Suppose there exists $i \varepsilon B2$ that effects $T23[i,(ml,ml)]$.

Let $\max_{i\varepsilon A} E_i(t_i) = a + 1$. Then at PC_a, by Theorem 3,

$s_i \neq S2[ml]$. Thus the first $i \varepsilon B2$ that effects $T23[ml,ml)]$ has a path to SINK at t_i (by Theorem 1). From all $i \varepsilon B2$ that effect $T23[t_i,i,(ml,ml)]$ while having a path to SINK, let q_o denote the node having the shortest path. Suppose the path is

$$Q = q_o \rightarrow q_1 \rightarrow \ldots q_k \rightarrow (SINK = q_{k+1}) .$$

By Theorem 1 all $q \varepsilon Q$ have $s_q(t_{q_o}) = S2[ml]$. But q_1 can only effect T21 or $T2\tilde{2}$, and q_1 cannot effect T21 unless receiving a message from q_o which cannot be sent because q_o does not effect T21. Hence q_1 will detect a failure of link (q_o,q_1) and by Lemma B.5 the proof is complete.

<u>Case 2</u>: Suppose there is no $i \varepsilon B2$ that effects $T23[i,(ml,ml)]$.

Let $q_o \varepsilon B2$ denote a node such that $d_{q_o}(t_{q_o}-) = \min_{i\varepsilon B2} d_i(t_i-)$, and suppose $p_{q_o}(t_{q_o}-) = q_1$. Node q_1 cannot effect T23 (definition of Case 2) and cannot effect T13 (violates the definition of q_o). Thus, q_1 detects a failure of link (q_o,q_1) and a REQ(ml) is generated.

If at any time this REQ(ml) enters a node at S2 or $\tilde{S2}$, then q.e.d. by Lemma B.5. Otherwise the REQ(ml) keeps moving through nodes at S1 having decreasing d_i. The REQ(ml) cannot be received by a node at S3 because this violates Case 2 or the definition of q_o. Since for all i, $d_i \geq 0$, d_i is an integral number and the only node with $d_i = 0$ is the SINK, the REQ(ml) will arrive at SINK after a finite number of steps. Q.e.d.

Lemma B.7

If a node i_o sends a REQ(ml) while $s_{i_o} = S1$, then a REQ(ml) arrived or will arrive at SINK in finite time.

Proof

By Lemma B.4, we have to prove only the case in which for all i, $n_i \leq ml$, and by Theorem 1, the REQ(ml) sent by i_o may encounter only nodes having $n_i = ml$.

If there exists a node i such that $s_i = S3[ml]$, then q.e.d. by Lemma B.6. Hence we may assume that for all i, $s_i \neq S3[ml]$ and therefore by Theorem 1 the REQ(ml) is in a tree rooted at SINK. Thus as in the proof of Lemma B.6, the REQ(ml) either arrives at a node in S2 or $\tilde{S2}$ (q.e.d. by Lemma B.5) or travels through nodes at S1, with decreasing d_i until it arrives at SINK, q.e.d.

Acknowledgement

The first stages of this work were performed when A. Segall was with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, and with the Codex Corporation, Newton, Mass., where he benefited from close collaboration with R.G. Gallager, G.D. Forney and S.G. Finn. Thanks are also due to Mr. M. Sidi for useful suggestions.

References

[1] A. Segall, The modeling of adaptive routing in data-communication networks, IEEE Trans. on Comm., Vol. COM-25, pp. 85-95, Jan. 1977.

[2] A. Segall and M. Sidi, Optimal failsafe distributed routing in data-communication networks, in preparation.

[3] G. Ludwig and R. Roy, Saturation routing network limits, Proc. IEEE, Vol. 65, No. 9, pp. 1353-1362, Sept. 1977.

[4] R.G. Gallager, A minimum delay routing algorithm using distributed computation, IEEE Trans. on Comm., Vol. COM-25, pp. 73-85, Jan. 1977.

[5] A. Segall, Optimal distributed routing for line-switched data networks, submitted to IEEE Trans. on Comm.

[6] W.D. Tajibnapis, A correctness proof of a topology information maintenance protocol for a distributed computer network, Communications ACM, Vol. 20, No. 7, pp. 477-485, July 1977.

[7] M. Schwartz, Computer-Communication Networks: Analysis and Design, Prentice-Hall, 1977.

[8] W.E. Naylor, A loop-free adaptive routing algorithm for packet switched networks, Proc. 4th Data Communication Symposium, Quebec City, pp. 7.9 - 7.14, Oct. 1975.

[9] R.G. Gallager, Loops in multicommodity flows, Paper ESL-P-772, M.I.T., Sept. 1977.

[10] G.V. Bochmann and J. Gecsei, "A unified method for the specification and verification of protocols", Publication #247, Departement d'Informatique, University of Montreal, Nov. 1976. To be presented at the IFIP-Congress 1977, Toronto.

[11] P.M. Merlin, A methodology for the design and implementation of communication protocols, IEEE Trans. on Communications, Vol. COM-24, No. 6, pp. 614-621, June 1976.

[12] C.A. Sunshine, Survey of communication protocol verification techniques, Trends and Applications 1976: Computer Networks, (Symposium sponsored by IEEE Computer Society; National Bureau of Standards), Gaithersburg, Maryland, Nov. 1976.

[13] M.G. Gouda and E.G. Manning, protocol machines: A concise formal model and its automatic implementation. Proceedings of the Third International Conference on Computer Communication, pp. 346-345, Toronto, Aug. 1976.

[14] S.G. Finn, Resynch network protocols, Proc. of ICC, 1977.

[15] R.G. Gallager, personal communication.

Footnote

1. The FACTS given in the algorithm are displayed for helping in its understanding and are proved in Theorem 2.

Table 1 - The Basic Algorithm

For MSG(d,ℓ)

$N_i(\ell) \leftarrow$ RCVD

$D_i(\ell) \leftarrow d + d_{i\ell}$;

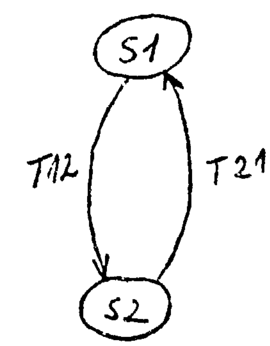
CT $\leftarrow 0$

Execute FINITE-STATE-MACHINE

BASIC-FINITE-STATE-MACHINE

$\boxed{p_i}$ $\boxed{d_i}$

ℓ	1	2	....	K_i
$D_i(\ell)$			....	
$N_i(\ell)$			....	



State S1

T12: Condition 12 MSG$(d, \ell = p_i)$, CT $= 0$.

Action 12 $d_i \leftarrow \min\limits_{k:N_i(k)=\text{RCVD}} D_i(k)$

transmit MSG(d_i) to all k s.t. $k \neq p_i$.

State S2

T21: Condition 21 $\forall k$, then $N_i(k) =$ RCVD.

Action 21 transmit MSG(d_i) to p_i;

$p_i \leftarrow k^*$ that achieves $\min\limits_{k} D_i(k)$;

$\forall k$, set $N_i(k) \leftarrow$ nil;

CT $\leftarrow 1$.

Table 2a - Variables of the Algorithm of Table 3.

Note: It is assumed that the network is composed by K nodes.

Variable Name	Meaning	Domain of Values
p_i	preferred neighbor	nil,1,2,...,K
d_i	estimated distance from SINK	∞,1,2,3,...
$d_{i\ell}$	estimated distance of link (i,ℓ)	1,2,3,...
n_i	current counter number	0,1,2,...
mx_i	largest number m received by node i	0,1,2,...
CT	control flag	0,1
$N_i(\ell)$	last number m received from ℓ after i completed last update cycle	nil,0,1,2,...
$D_i(\ell)$	$d + d_{i\ell}$ for last d received from ℓ	∞,1,2,...
$F_i(\ell)$	status of link (i,ℓ)	DOWN,READY,UP
$z_i(\ell)$	synchronization number used by i to bring link (i,ℓ) UP	0,1,2,...

Table 2b - Messages received by the algorithm of Table 3.

Message Format	Meaning	Domain of Values
$MSG(m,d,\ell)$	updating message from ℓ	m = 0,1,2,... d = ∞,0,1,2,... ℓ = 1,2,...,K
$FAIL(\ell)$	failure detected on link (i,ℓ)	ℓ = 1,2,...,K
$WAKE(\ell)$	link (i,ℓ) becomes operational	ℓ = 1,2,...,K
REQ(m)	request for new update cycle with $n_{SINK} > m$	m = 0,1,2,...

Table 3 - Algorithm for an Arbitrary Node i

I.1 For REQ(m)

if $p_i \neq$ nil, then send REQ(m) to p_i.

I.2 For FAIL(ℓ)

I.2.1 $F_i(\ell) \leftarrow$ DOWN;

I.2.2 CT $\leftarrow$ 0;

I.2.3 Execute FINITE-STATE MACHINE;

I.2.4 if $p_i \neq$ nil, then send REQ(n_i) to p_i.

I.3 For MSG(m,d,ℓ)

I.3.1 if $F_i(\ell)$ = READY, then $F_i(\ell) \leftarrow$ UP

(Fact[1]: $m > z_i(\ell)$);

I.3.2 $N_i(\ell) \leftarrow m$;

I.3.3 $D_i(\ell) \leftarrow d + d_{i\ell}$;

I.3.4 $mx_i \leftarrow \max\{m, mx_i\}$;

I.3.5 CT $\leftarrow$ 0;

I.3.6 Execute FINITE-STATE MACHINE.

I.4 For WAKE(ℓ)

(Fact : $F_i(\ell)$ = DOWN)

wait for end of WAKE synchronization (see Section 2.7);

if WAKE synchronization is successful, then

$z_i(\ell) \leftarrow \max\{n_i, n_\ell\}$;

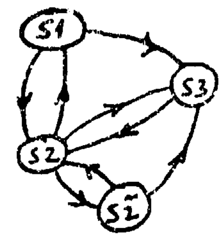
$F_i(\ell) \leftarrow$ READY;

$N_i(\ell) \leftarrow$ nil;

if $p_i \neq$ nil, then send REQ($z_i(\ell)$) to p_i.

(continued)

Table 3 (cont'd)



II. FINITE STATE MACHINE

State S1

II.1.1 T12 Condition 12 $MSG(m = mx_i, d \neq \infty, \ell = p_i)$, $CT = 0$,

II.1.2 Fact 12 $m \geq n_i$

II.1.3 Action 12 $d_i \leftarrow \min\limits_{\substack{k:F_i(k) = UP \\ N_i(k) = m}} D_i(k);$

II.1.4 $n_i \leftarrow m;$

II.1.5 $\forall k$ s.t. $F_i(k) = READY$ if $n_i > z_i(k)$, then

$F_i(k) \leftarrow UP$, $N_i(\ell) \leftarrow nil;$

II.1.6 transmit (n_i, d_i) to all k s.t. $F_i(k) = UP$

and $k \neq p_i$;

II.1.7 $CT \leftarrow 1$.

II.2.1 T13 Condition 13 $(MSG(\ell = p_i, d = \infty, m)$ or $FAIL(\ell = p_i))$, $CT = 0$.

II.2.2 Fact 13 If MSG, then $m \geq n_i$.

II.2.3 Action 13 $d_i \leftarrow \infty;$

II.2.4 if MSG, then $n_i \leftarrow m;$

II.2.5 $\forall k$ s.t. $F_i(k) = READY$, if $n_i > z_i(k)$, then

$F_i(k) \leftarrow UP$, $N_i(k) \leftarrow nil;$

II.2.6 transmit (n_i, d_i) to all k s.t. $F_i(k) = UP$

and $k \neq p_i$;

II.2.7 $p_i \leftarrow nil;$

II.2.8 $CT \leftarrow 1$.

(continued)

Table 3 (cont'd)

	State S2		
II.3.1	T21	Condition 21	$\forall k$ s.t. $F_i(k) = UP$, then $N_i(k) = n_i = mx_i$;
II.3.2			$\exists k$ s.t. $F_i(k) = UP$ and $D_i(k) \leq d_i$;
II.3.3			if $CT = 0$, then MSG;
II.3.4			$D_i(p_i) \neq \infty$.
II.3.5		Fact 21	$d_i \neq \infty$, $p_i \neq$ nil.
II.3.6		Action 21	Transmit (n_i, d_i) to p_i;
II.3.7			$p_i \leftarrow k^*$ that achieves $\min\limits_{k:F_i(k)=UP} D_i(k)$;
II.3.8			$\forall k$ s.t. $F_i(k) = UP$, set $N_i(k) \leftarrow$ nil;
II.3.9			$CT \leftarrow 1$.
II.4.1	T22	Condition 22	MSG$(m = mx_i > n_i,\ d \neq \infty, \ell = p_i)$, $CT = 0$.
II.4.2		Action 22	Same as Action 12.
II.5.1	T2$\tilde{2}$	Condition 2$\tilde{2}$	FAIL$(\ell \neq p_i)$, $CT = 0$.
II.5.2		Action 2$\tilde{2}$	$CT \leftarrow 1$.
II.6.1	T23	Condition 23	Same as Condition 13.
II.6.2		Fact 23	Same as Fact 13.
II.6.3		Action 23	Same as Action 13.
	State S3		
II.7.1	T32	Condition 32	$\exists k$ s.t. $F_i(k) = UP, mx_i = N_i(k) > n_i, D_i(k) \neq \infty$.
II.7.2		Fact 32	$p_i =$ nil, $d_i = \infty$.
II.7.3		Action 32	Let k^* achieve $\min\limits_{\substack{k:F_i(k)=UP \\ N_i(k)=mx_i}} D_i(k)$.

(continued)

Table 3 (cont'd)

II.7.4			Then $p_i \leftarrow k^*$;
II.7.5			$n_i \leftarrow mx_i$;
II.7.6			$d_i \leftarrow D_i(k^*)$;
II.7.7			$\forall k$ s.t. $F_i(k) = \text{READY}$, if $n_i > z_i(k)$, then $F_i(k) \leftarrow \text{UP}$, $N_i(k) \leftarrow \text{nil}$;
II.7.8			transmit (n_i, d_i) to all k s.t. $F_i(k) = \text{UP}$ and $k \neq p_i$;
II.7.9			$CT \leftarrow 1$.
	State $\tilde{S2}$		
II.8.1	$T\tilde{2}2$	Condition $\tilde{2}2$	$MSG(m = mx_i > n_i, d \neq \infty, \ell = p_i)$, $CT = 0$.
II.8.2		Action $\tilde{2}2$	Same as Action 12
II.9.1	$T\tilde{2}3$	Condition $\tilde{2}3$	Same as Condition 13
II.9.2		Fact $\tilde{2}3$	Same as Fact 13
II.9.3		Action $\tilde{2}3$	Same as Action 13 .

Table 4

The Algorithm for the SINK

For REQ(m)

CT ← 0 ;

execute FINITE-STATE-MACHINE.

For FAIL(ℓ)

$F_i(\ell)$ ← DOWN ;

CT ← 0 ;

execute FINITE-STATE-MACHINE .

For MSG(m,d,ℓ)

$N_i(\ell)$ ← m ;

CT ← 0;

execute FINITE-STATE-MACHINE .

For WAKE(ℓ)

(Fact: $F_i(\ell)$ = DOWN)

wait for end of WAKE synchronization;

if WAKE synchronization is successful, then

$F_i(\ell)$ ← READY;

CT ← 0 ;

execute FINITE-STATE-MACHINE.

For START

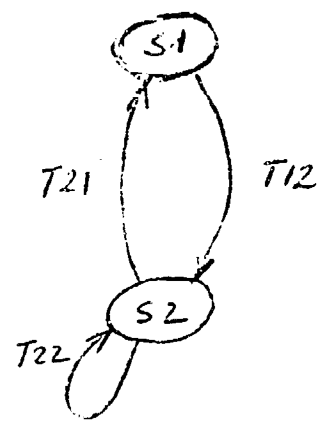
CT ← 0;

execute FINITE-STATE-MACHINE .

(continued)

Table 4 (cont'd)

FINITE-STATE MACHINE FOR SINK



State S1

T12 Condition 12 (CT = 0) and (REQ($m = n_{SINK}$) or FAIL or WAKE or START)

Action 12 if (REQ or FAIL or WAKE), then $n_{SINK} \leftarrow n_{SINK} + 1$;

$\forall k$ s.t. $F_i(k)$ = READY, then $F_i(k) \leftarrow$ UP, $N_i(k) \leftarrow$ nil;

transmit $(n_{SINK}, 0)$ to all k s.t. $F_i(k)$ = UP;

CT $\leftarrow$ 1.

State S2

T21 Condition 21 $\forall k$ s.t. $F_i(k)$ = UP, then $N_i(k) = n_{SINK}$;

MSG or START.

Action 21 $\forall k$ s.t. $F_i(k)$ = UP, then $N_i(k) \leftarrow$ NIL;

CT $\leftarrow$ 1.

T22 Condition 22 (CT = 0) and (REQ($m = n_{SINK}$) or FAIL or WAKE)

Action 22 Same as Action 12.

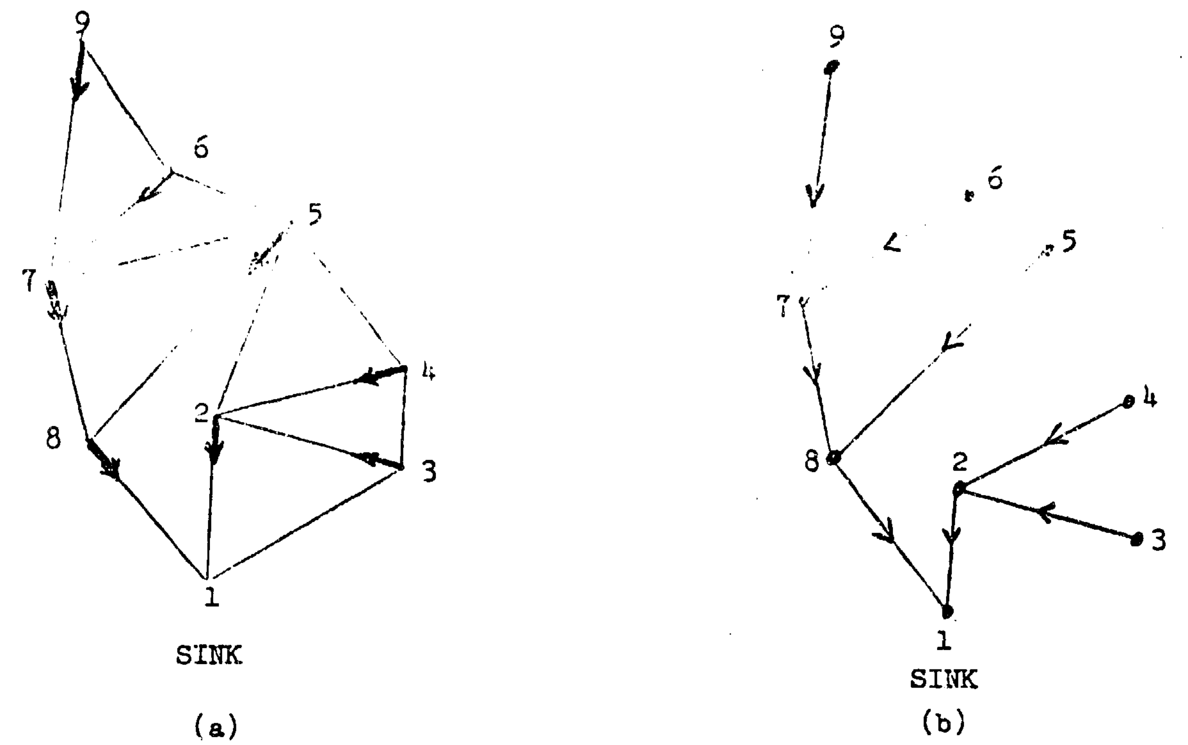


Fig. 1: (a) Network example
(b) Corresponding directed tree

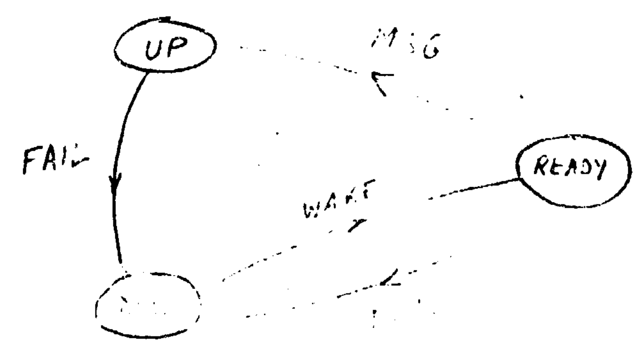


Fig. 2: Possible changes of $F_i(\ell)$

Distribution List

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 Copies
ssistant Chief for Technology Office of Naval Research, Code 200 Arlington, Virginia 22217	1 Copy
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 Copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 Copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 Copy
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, California 91106	1 Copy
New York Area Office (ONR) 715 Broadway - 5th Floor New York, New York 10003	1 Copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 Copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 Copy

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 NAICOM/MIS Planning Branch (OP-916D) Office of Chief of Naval Operations Washington, D.C. 20350	1 Copy
Mr. Kin B. Thompson Technical Director Information Systems Division (OP-91T) Office of Chief of Naval Operations Washington, D.C. 20350	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
Captain Richard L. Martin, USN Commanding Officer USS Francis Marion (LPA-249) FPO New York 09501	1 Copy