July 1987

LIDS-P-1543
(revised)

Broadcasting Topology Information in Computer Networks*

John M. Spinelli Robert G. Gallager

Massachusetts Institute of Technology

Cambridge, Massachusetts 02139

Revised: June, 1987

Abstract

An algorithm is presented which allows each node in a computer network to maintain a correct view of the network topology despite link and node failures. Reliability is achieved without transmitting any information other than the operational status of links. Messages are only sent in response to topological changes: periodic retransmission is not required.

*This research was conducted at the MIT Laboratory for Information and Decision Systems, and the Center for Intelligent Control Systems. It was supported in part by a National Science Foundation Graduate Fellowship, by the National Science Foundation under Grant NSF-ECS-8310698, by the Defense Advanced Research Projects Agency under Grant ONR/N00014-84-K-0357, and by the Army Research Office under Grant ARO DAAL03-86-K-0171.

1 Introduction

At any time while a store and forward computer network is operating, one or more of its communication links or processing nodes may malfunction or be put back into service. The recovery of the network from such a change in its topology is an essential part of providing reliable data communication. This paper is concerned with the problem of keeping each network node informed of the entire network topology when that topology occasionally changes over time. Any network which uses decentralized adaptive routing needs to address this problem. The classic example of this is the ARPANET, where each node maintains a map of the entire network and uses it in making routing decisions [8]. Even networks which use some form of hierarchical routing need to solve this problem within some level of the hierarchy.

Topological changes may occur at any time. Since all messages sent in the network are subject to delay, a node can never be certain that it knows the correct topology at some instant of time. However, distributed algorithms can be designed to guarantee that each node is made aware of the correct status of each link to which it has a physical path, provided that the topology does not change for a sufficient but finite time. Algorithms which accomplish this are called topology algorithms.

A topology algorithm is a set of rules governing the topology information stored at a node, as well as the contents, transmission, and reception of algorithm messages. These messages are called *topology updates* or *update messages.* They usually contain an indication of the operational status (up or down) of one or more network links. When a network is started or reinitialized, the algorithm must determine the initial network topology and communicate it to every node. Thereafter, the topology

information must be kept current by update messages which are sent periodically or in response to a topological change.

Although it might appear that any effective method of broadcast could serve as a topology algorithm, there are several subtle issues which cause difficulties.

1. The links which are used to communicate topology information are themselves subject to fail at any time. The failure of a link creates additional topology information which must be communicated. It may also block the transmission of previous topology information about other links, and in the worst case may divide the network into two disconnected sets of nodes.

2. A link may experience several topology changes within a short time. Other network nodes must eventually determine which change was the most recent. In general, nodes must be able to distinguish between old and new information about the status of a link.

3. While a topology algorithm is running, additional topology changes may occur. The topology algorithm must be capable of either incorporating new information during execution, or of starting a new algorithm version. If different versions are used, each node must be able to determine which is the most recent version, a problem similar to 2 above.

4. The repair of a single link can cause two parts of the network which were disconnected to reconnect. Each part may have arbitrarily out-of-date topology information about the other. The algorithm must ensure that the two parts eventually agree, and adopt the correct network topology.

Several algorithms have been designed in order to overcome these difficulties

[1,4,5,7,8,9,11,12,13,14,15,16]. The problem of broadcasting topology changes has been formulated in many different ways, and is often solved as part of a distributed routing algorithm. A common approach is to include auxiliary information such as message counters, sequence numbers, or age fields in update messages along with the topology information itself. The sequence numbers are used to distinguish between old and new information and to stop the flooding of messages. Timers at the nodes may be used to enforce some minimum or maximum time interval between the transmission of topology information. Age fields or time stamps may be also be used as auxiliary information in algorithm messages to ensure that old topology information is eventually deleted. The ARPANET algorithm [8] uses a combination of sequence numbers, age fields, and timers. It also requires periodic retransmission of topology information. Perlman [9] presents improvements to this algorithm which, among other things, make it less timer dependent. Other algorithms [1,4,5,7,11,12,14,15,16] do not use timers, age fields, or time stamps, and send messages only in response to receiving other messages, or in response to topological changes in adjacent links. We call this class of algorithm event driven. Such algorithms may still rely on bounded message counters to distinguish between old and new information. They may respond to topological changes by rebuilding the entire network topology as in [4], or by modifying an existing topology to reflect the change as in [11]. Several different types of messages are sometimes used for broadcasting information, collecting acknowledgements, and terminating the algorithm.

The four issues mentioned earlier can make it quite difficult to prove the correctness of topology algorithms, especially when the algorithms are complex. For

example, it was recently shown in [14] that the topology algorithms in [4] and [12] can fail to operate properly in some unusual circumstances. The problem with the algorithm in [4] went unnoticed for nearly ten years. This argues for topology algorithms which can be easily shown to be correct.

In this paper we take a rather unconventional approach to solving the topology problem. The algorithm which we present, called the *Shortest Path Topology Algorithm* (SPTA), uses no auxiliary information at all. The update messages consist only of topological information, i.e. link status information. The algorithm is purely event driven in that nodes transmit messages only in response to receiving a topology update message from a neighbor, or to detecting a status change in an adjacent link. It does not rely on periodic retransmission of messages, or use timers, counters, or clocks of any kind. The update messages are used by a node to modify its existing topology, and there are no special cases for reconnection of disconnected parts of the network. The simple message structure allows a theoretical proof of correctness that is rather straightforward.

There are two main motivations for constructing an algorithm with the above characteristics. Although the use of auxiliary information is a logical way to deal with the four difficulties mentioned earlier, it also introduces additional problems [9]. For example, if sequence numbers are used, the finite bit field used to store them may eventually wrap around. While this can be avoided by choosing a large bit field, some provisions should still be made for resetting the numbers [14]. The introduction of auxiliary information into update messages usually leads to increasing complexity. An algorithm which avoids auxiliary information entirely avoids also the complexities and difficulties associated with it. Apart from this, there is

the academic question of whether topological information alone is sufficient to solve the topology problem. SPTA shows that although auxiliary information may be desirable in some circumstances, it is not required in topology algorithms. By exchanging only link status information, the network nodes can arrive at and maintain a correct topology.

When a single link changes operational status, the performance of SPTA is similar to traditional approaches such as the ARPANET update algorithm. However, when many links change status in a very short time, SPTA can send a large number of messages in worst case situations. These performance issues are addressed in section 3.

2 The SPTA Algorithm

The main idea behind SPTA is that each node is viewed as trying to construct the network topology based on reliable information that it has about the status of adjacent links, and possibly unreliable or inconsistent information (in the sense that it might be outdated) that it has received from neighboring nodes. When a node receives contradictory information about the status of a link from two or more of its neighbors, it resolves the conflict by 'believing' the neighbor which it calculates to be closest in hops to the link in question. While hop length is not a unique metric for resolving such conflicts, it results in a fairly simple algorithm.

2.1 Assumptions

The correct operation of any topology algorithm depends strongly on the way in which link status changes (failures and repairs) are detected by the network nodes. Each link is considered to be bidirectional. There is a data link control protocol operating at each end node of a link which decides whether the link is *up* (operating) or *down* (not operating). Due to communication delay, the decisions made by the two end nodes of a link may be nonsimultaneous. The following assumptions are made about the operation of the data link control protocol.

A1) When a link is down at one end node, it must eventually be called down at the other end node, before either end node can call it up again.

A2) If a link is called up at one end node, then within finite time, either the opposite end node must call it up or the first must call it down again.

A3) If a message sent by a node i on link (i,j) does not arrive correctly at node j within a finite time, then link (i,j) will be called down by both i and j in a finite time.

A4) Links preserve the order of transmitted messages.

The following assumptions are made about the operation of the network nodes.

A5) A node failure is represented by the (perhaps nonsimultaneous) failure of the links adjacent to the node.

A6) While they are operating, nodes maintain the integrity of the data and messages stored in their memory.

Data link control protocols for achieving assumptions A1 thorough A4 are non-trivial. Examples of such protocols may be found in [2]. Section 3.1 discusses the operation of SPTA when the above assumptions are violated.

2.2 SPTA Data Structures and Rules

Each node i in the network maintains a topology table T^i called its main topology table. A topology table is a list of the operational status of each link in the network. We refer to the single bidirectional link between nodes m and n as either (m,n) or (n,m), which ever is more convenient. T^i contains an entry $T^i(m,n)$ for each link (m,n), and reflects node i's current best estimate of the network topology. It is the official topology that would be used by a routing algorithm operating at node i. In addition to its main topology table, node i maintains a port topology table T^i_j associated with each neighboring node j. The entry in this table for link (m,n) is denoted $T^i_j(m,n)$. The information stored in table T^i_j is the latest topology information received by node i from node j. The tables stored at node i are shown in Figure 1. When a node's main topology table changes, it sends a message notifying each of its neighbors of the change. Therefore, T^i_j is merely a delayed version of node j's main topology table, T^j.

Most algorithm messages consist of a single link name, (m,n) together with the link's status (up or down). However, when a link becomes operational, its end nodes exchange their entire main topology tables. The contents of a table is sent as a single message.

SPTA consists of a set of rules for sending messages and updating the topology tables described above. The following rules are followed by each network node.

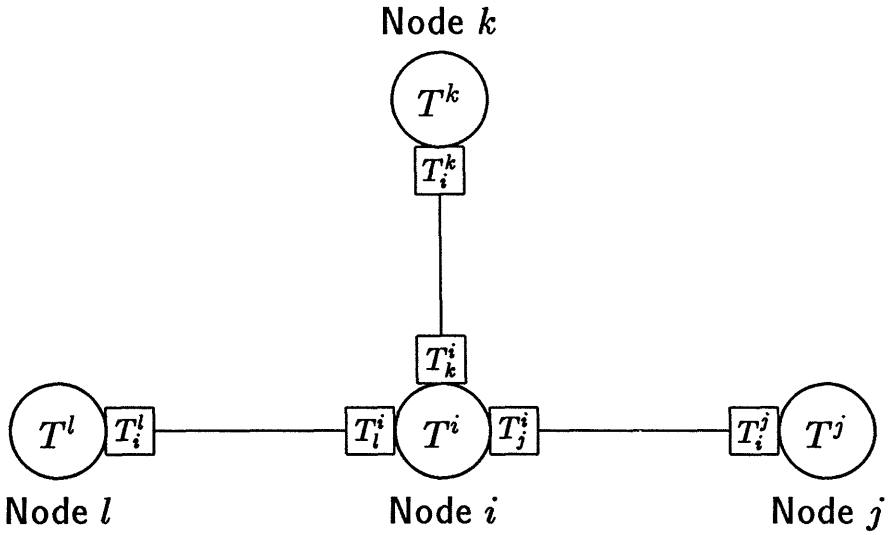

Figure 1: The Main and Port Topology Tables for a Simple Network.

Communication Rules

R1) When a link status entry in a node's main topology table changes, a message containing the new value is sent on each operating adjacent link.

R2) When the link protocol at a node detects that an adjacent link has become operational, the node transmits its entire main topology table over that link.

Topology Table Update Rules

R3) When the link protocol at a node detects that an adjacent link has failed, the failed status is entered in the node's main topology table.

R4) When a node receives a single link status message from a neighbor, it enters the change in the port topology table associated with that neighbor.

R5) When a node receives an entire main topology table from a neighbor, it enters the changes in the port topology table associated with that neighbor. It also

lists the link on which the message was received as *up* in its main topology table.

R6) When the application of rules R3, R4, or R5 causes an entry in a node's main or port topology tables to change, the node updates its main topology table by using the *main topology update algorithm* described below.

2.3 Main Topology Table Update Algorithm

The following algorithm is used by each node i to construct its main topology table T^i based on its knowledge of the status of adjacent links, and the information stored in its port topology tables. It consists of iterations which are very similar to those of Dijkstra's shortest path algorithm [3] when all the links are taken to have a length of 1. The following variables are used by the algorithm by each node i:

P_k: (for $k \geq 1$) the set of nodes whose shortest hop path to node i has k links, using only links which are *up* in topology T^i.

L_k: (for $k \geq 1$) the set of links (m, n) such that the shortest hop path from node i to the closer end node of (m, n) has k links, using only links which are *up* in topology T^i.

$N(m)$: a neighbor of node i that is the first node on a shortest hop path from node i to node m, using only links which are up in topology T^i. $N(m)$ is referred to as the *label* of node m.

$s(i, m)$: node i's current operational status for adjacent link (i, m). This is provided by the data link control algorithm operating at node i.

The purpose of the k^{th} iteration of the algorithm is to enter into T^i the status of those links contained in L_k. This is done by selecting, for each link $(m,n) \in L_k$, a port topology table to believe concerning link (m,n)'s status. The status entry for link (m,n) in this port table is then entered in T^i.

At the start of the 1^{st} iteration of the algorithm. T^i contains the status of each link adjacent to i. For completeness, we define $P_0 = \{i\}$. P_1 is the set of neighbors of i connected by working links. In general, at the start of the k^{th} iteration the sets P_k and P_{k-1} have already been defined. The set L_k can be constructed from P_k and P_{k-1} by chosing those links which have at least one end node in P_k, but no end node in P_{k-1}. For each link $(m,n) \in L_k$, we chose an end node m such that $m \in P_k$. Then $N(m)$ is a neighbor of i which lies on a shortest hop path from i to link (m,n). The port topology table associated with this neighbor is the one that will be believed concerning the status of link (m,n). If link (m,n) is *up* and $n \notin P_k$ then $n \in P_{k+1}$, and m is on a shortest path from i to n. We can set $N(n)$ equal to $N(m)$, and construct the set P_{k+1} in preparation for the $(k+1)^{\text{st}}$ iteration. The algorithm terminates when $P_k = \emptyset$. This indicates that the status of each link connected to node i has been entered in T^i.[1]

In the following more formal presentation of the main topology update algorithm, all of the sets P_k are assumed to be initially empty.

Main Topology Update Algorithm at node i

$$T^i(m,n) := \begin{cases} s(m,n) & \text{for each link adjacent to } i \\ down & \text{for other links} \end{cases}$$

[1] A link l is connected to a node i if there is a path of operating links connecting i with one of the end nodes of l.

$P_0 := \{i\}$

$P_1 := \{n \mid T^i(i,n) = up\}$

for each $n \mid n \in P_1$

$N(n) := n$

$k := 1$

do while $P_k \neq \emptyset$

begin

$L_k := \{(m,n) \mid m \notin P_{k-1}$ **and** $n \notin P_{k-1}$ **and** $(m \in P_k$ **or** $n \in P_k)\}$

for each link $(m,n) \in L_k$

begin

(assume without loss of generality that $m \in P_k$)

$T^i(m,n) := T^i_{N(m)}(m,n)$

if $T^i(m,n) =$ up **and** $n \notin P_k$ **and** $n \notin P_{k+1}$

then $N(n) := N(m)$; $P_{k+1} := P_{k+1} \cup \{n\}$

end

$k := k + 1$

end

stop

Notice that when an adjacent link (i,j) fails, the algorithm at node i does not include node j in the set P_1. Because of this, for each node m, $N(m) \neq j$. Therefore, node i effectively disregards all information which is stored in the port topology table T^i_j.

When two or more neighbors of i are at a minimum hop distance k from some link $(m,n) \in L_k$, the algorithm must choose which neighbor to believe concerning

link (m,n)'s status. The way in which ties are resolved depends upon the order in which the members of L_k are processed, and upon the end node of (m,n) which is selected if both m and n are in P_k. The tie breaking rule used does not affect the correctness of SPTA.

2.4 Correctness Proof of SPTA

Assume that at some time t_s a network is in steady state. This means that for each node within a connected component of the network, the main topology tables are correct for each link which is adjacent to a member of the component. In addition, no algorithm messages are being transmitted. Note that a node is in steady state immediately after being reinitialized. Between time t_s and some later time t_0 an arbitrary number of link topology changes occur. For a sufficient but finite time interval after t_0 assume that no further topology changes occur. The required length of this interval will be addressed in section 3. We wish to show that at some later time $t_f \geq t_0$ steady state has been reestablished. Let T^* be the correct network topology that an omniscient observer would see upon examining the network after t_0. We say that node i "knows the correct topology" if its main topology T^i agrees with T^* for all links that are connected to i. Node j is called an *active neighbor* of node i if link (i,j) is operating according to T^*. We begin by showing the following theorem.

Theorem 1. *SPTA works correctly in the sense that, under the preceding assumptions, there is a finite time t_f after which each node knows the correct topology.*

Proof. In what follows, we say that a link l is at distance n away from i if in the graph defined by T^* the shortest path from i to the closest end node of l is n

hops long. We will show by induction that for each integer $n \geq 0$ there is a time $t_n \geq t_s$ after which, for each node i, T^i agrees with T^* for all links that are at a distance of n or less from i. The induction hypothesis is clearly true for $n = 0$ since each node i knows the correct status of its adjacent links and records them in its main topology table T^i. We first establish the following lemma.

Lemma 1. *Assume that the induction hypothesis is true for time t_n. Then there is a time $t'_{n+1} \geq t_n$ after which the port topology table T^i_j, for each active neighbor j of node i, agrees with T^* for each link at a distance n or less from j.*

Proof. Consider waiting a sufficient time after t_n for all messages which were sent from j to i before t_n to arrive. By rules R1, R2, R4, and R5 of the algorithm, T^i_j agrees with T^j for all links which are not adjacent to i. Therefore, by the induction hypothesis T^i_j agrees with T^* for each link at a distance of n or less from j which is not adjacent to i. By rules R3 and R5 the correct status of links adjacent to i are recorded in T^i_j. Therefore, T^i_j also agrees with T^* for all links adjacent to i. This proves the lemma.

To complete the proof of Theorem 1 we must show that there is a time $t_{n+1} > t'_{n+1}$ such that for all $t \geq t_{n+1}$ and nodes i, T^i agrees with T^* for each link l which is at a distance $n+1$ from i. Consider the first time that link l is processed by the main topology update algorithm after the conditions of Lemma 1 are satisfied. Then, link l will belong to the set L_{n+1}. Also, the closest end node of l to node i will belong to the set P_{n+1}, and will have a label which is one of the active neighbors of i (say j) that is at distance n from l. By Lemma 1, the entry of T^i_j for link l, which will be copied into T^i when link l is processed, will agree with the corresponding entry in T^*. Since Lemma 1 holds for all time $t \geq t'_{n+1}$, the entry for link l in T^i

will be correct at all subsequent times when link l will be processed by the main topology update algorithm. This completes the proof of Theorem 1.

After t_f a node i's main topology table will not change. Since a node only transmits messages when its main topology changes, node i will not transmit any messages after t_f. A finite time later a condition of steady state will be reestablished. This completes the correctness proof of SPTA.

3 Algorithm Characteristics

There are several criteria which can be used to evaluate the efficiency of a topology algorithm. These include the number of messages which the algorithm sends, the amount of time which it takes to terminate, the amount of processing which is required at the network nodes, and the amount of node memory which the algorithm and its data consume. The performance of an algorithm with regard to the first three of these criteria is usually dependent on the particular topology change event under consideration. By far the most important event is a status change in a single link. Since links which do not have a common end node can be expected to change status somewhat independently, the probability of many nearly simultaneous link status changes is very small. Therefore, the efficiency of topology algorithms when many links change status is a secondary consideration. A node status change can cause nearly simultaneous status changes in the links adjacent to the node, but since most networks are sparse this typically involves only a small number of links. In the remainder of this section we consider the efficiency of SPTA, when operating on a network of N nodes and L links, with an emphasis on single link status changes.

To evaluate the amount of time which SPTA takes to run, we introduce the

notion of *time complexity* described in [6]. The time complexity of an algorithm is the number of units of time required if each communication of a message over a link requires at most one time unit and computation requires negligible time. We are interested in the amount of time that the algorithm requires to terminate following a set of status changes involving K links. It is shown in Appendix A that, subject to a few assumptions on the operation of link transmission queues, the time complexity, T, for SPTA is $O(N+K)$, and in fact $T < 2(N+K)$. Since the final status of each of K links must be communicated across the diameter of the network, this is an optimal result for the order of T. For single link topology changes, this is the same result as for the ARPANET update algorithm[8].

In the correctness proof of SPTA it was assumed that no topology changes occurred for a "sufficient but finite time." It can be seen that the algorithm terminates in a time which is roughly equivalent to the message propagation time across the network. Assuming that no status changes occur during this time is equivalent to assuming that the average time between status changes is much larger then the message propagation time across the network.

The communication complexity of an algorithm is the sum, over all network links, of the number of messages sent on each link. For single link topology changes, this can be established by examining rule R1 of SPTA. When a node's main topology table changes, it sends a message on each of its adjacent links. This results in $2L$ messages being sent on an L link network and gives $O(L)$ communication complexity. This is the same result as for the ARPANET update algorithm.

When multiple link status changes occur over a short period of time, the calculation of communication complexity for SPTA is complicated. A general bound

on communication will not be presented, but it is shown by example in Appendix B that in some situations the number of messages sent by a node can grow exponentially in the number of status changes. Such behavior is clearly undesirable, but it can be obtained only by very carefully choosing the message and status change timing. In practical situations, the ability to remove obsolete messages from queues, the low probability of many nearly simultaneous status changes, and the stochastic variation in message transmission time would tend to reduce the number of messages sent.

The amount of processing required by the algorithm can be calculated by examining the main topology update algorithm. Since each link is processed exactly once by this algorithm, its computational requirement is $O(L)$. As stated in section 2.2, a node runs the main topology update algorithm each time a message is received, or a status change is detected in an adjacent link. In many situations this is unnecessary, since the algorithm only modifies the main topology table when status information is received over a shortest hop path. For the case of a single link topology change, if the main topology update algorithm is only run when a message arrives on a shortest path (or when an adjacent link changes status) then each node runs the algorithm exactly once. This gives $O(L)$ computations per node, and is similar to the requirements of the ARPANET update procedure. The methods described in [8] for reducing the computational requirements of the ARPANET algorithm by maintaining a shortest path tree can also be applied to the main topology update algorithm of SPTA.

The memory requirement of SPTA at each node i is $O(LB_i)$ where B_i is the number of neighbors of node i. B_i is usually a small integer, but can be as large as

$N-1$ for a fully connected network. SPTA requires more memory than algorithms such as the ARPANET update procedure, due to its use of multiple topology tables at each node.

3.1 Fault Tolerance

In practice, it is difficult to guarantee that the assumptions made in section 2.1 are always valid. For example, errors may occasionally occur in a node's memory, violating A6. Like all event driven algorithms, SPTA has no ability to recover from undetected database errors. Without periodic retransmission, an undetected database error can persist for an arbitrary amount of time. Another possibility is for one or more nodes to temporarily violate the assumptions, and behave in an unpredictable manner. Such a situation has occurred on the ARPANET [10]. Once the nodes begin to faithfully execute SPTA once again, it is desirable for a condition of steady state to reestablish itself.

Although SPTA does not rely on periodic retransmission for correct operation under assumptions A1 through A6, periodic retransmission can be used to provide recovery from situations where the operating assumptions are temporarily violated. Consider augmenting SPTA such that each node periodically transmits its entire main topology table to each active neighbor. Let t_s be a time after which the assumptions A1 through A6 are valid and no further topology changes occur. A finite time after t_s each node i will successfully transmit its main topology table T^i to each active neighboring node j. This implies that T^i and T_i^j will be consistent. This fact combined with a node's knowledge of the status of its adjacent links is sufficient to prove Lemma 1. The rest of the correctness proof of SPTA follows as

before. Therefore, when periodic retransmission is used, SPTA will converge to the correct topology a finite time after its operating assumptions are valid. Because it does not use sequence numbers or age fields, SPTA is immune to the type of problem which has occurred on the ARPANET [10].

Appendix A: Time Complexity

Consider some sequence of status changes involving at most K links. Each of the K links may change status one or more times at either cr both end nodes. We wish to show that if all status changes cease by time 0, then all nodes will know the correct topology by time $2(K + N)$. To obtain this result, we make the following assumptions:

1. The links of the network are numbered consecutively from 1 to L, and this numbering is known to all nodes.

2. If a node i has several status changes to send to a neighbor, it sends the status changes for the closest (in hops) links to i first. For links at the same hop distance, it sends the status change for the lowest numbered link first. (Link hop distances are available to a node from the main topology update algorithm.)

3. If a link changes status in a node's main topology table before the old status has been sent on a neighboring link, then the old status is deleted from the queue of messages waiting to be sent.

4. If node i starts to send a status message to neighbor j at time t, j will have received it by time $t+1$.

5. For convenience, the transmission of a nodes main topology table over a link when it first comes up is considered as part of the link initialization procedure.

Let $h(i,l)$ be the hop distance of link l from node i in the final topology T^*. At each node i, we assign each link l that has changed status a distinct *index number*, $m(i,l)$, from the set $\{1,2,\ldots K\}$. The links are indexed in order of increasing hop distance, $h(i,l)$. Links at the same hop distance are indexed in order of increasing link number.

Lemma A1. *If j is a neighbor of i, and j is on a shortest path from i to link l, where l is a link that has changed status, then $m(j,l) \leq m(i,l)$.*

Proof. Let l' be any other link that has changed status for which $m(j,l') < m(j,l)$. We show first that

$$m(i,l') < m(i,l) \qquad (A1)$$

There are two cases to consider:

Case 1: $h(j,l') = h(j,l)$.

Then the link number of l' is less than the link number of l. We have

$$\begin{aligned} h(i,l') &\leq h(j,l') + 1 \\ &= h(j,l) + 1 \\ &= h(i,l) \end{aligned}$$

so (A1) is satisfied.

Case 2: $h(j,l') < h(j,l)$.

We have

$$\begin{aligned} h(i,l') &\leq h(j,l')+1 \\ &< h(j,l)+1 \\ &= h(i,l) \end{aligned}$$

so (A1) is satisfied again.

Now observe that there are $m(j,l)-1$ choices of l' for which $m(j,l') < m(j,l)$. For each of these, $m(i,l') < m(i,l)$. Since the numbers $m(i,l')$ are all distinct, $m(i,l) \geq m(j,l)$. Q.E.D.

Now define

$$t(i,l) = 2[m(i,l) + h(i,l)] \qquad (A2)$$

Theorem A1. *By time $t(i,l)$ and beyond, node i has the correct status for link l in its main topology table, and has transmitted that status to all its neighbors.*

Proof. We use induction on the value of $t(i,l)$. For the basis of the induction, note that $m(i,l) + h(i,l) \geq 1$ for all i and l. Consider any node i which has an adjacent link status change. If l is the lowest numbered link adjacent to i that changes status, then $m(i,l) + h(i,l) = 1$. By time 1, whatever status change was being sent at time 0 is completed, and node i will start to send the final status change for link l to all neighbors. By time 2, this status change will have been transmitted.

Now consider an arbitrary i,l for which $t(i,l) = t$. By the induction hypothesis, we assume that the theorem is valid for all j,l' such that $t(j,l') \leq t-2$. Consider each neighbor j of i that is on a shortest hop path to l. By lemma A1, $m(j,l) \leq m(i,l)$,

and by definition $h(j,l) = h(i,l) - 1$. Therefore, $t(j,l) \leq t - 2$. Together with the induction hypothesis, this implies that, by time $t - 2$, node i has received the final status for link l from every neighbor j which is on a shortest path to l. Also by time $t - 2$, node i will have the final status for link l entered in its main topology table.

To complete the proof we must show that node i will be able to transmit the status of link l to each neighbor by time t. Note that i has the correct topology by $t - 2$ for all l' such that $m(i,l') < m(i,l)$ (since $m(i,l') < m(i,l)$ implies that $h(i,l') \leq h(i,l)$). By time $t - 2$, i has transmitted the final status changes for all such l'. Therefore, no matter what node i is transmitting at time $t - 2$, it can start transmitting the final status change for l by time $t - 1$, and all neighbors will have received it by time t. Q.E.D.

Appendix B: Communication Complexity Example

We show by example that there are unusual situations where the number of messages sent by the algorithm can grow exponentially in the number of status changes. Figure 2 illustrates one such situation involving 2 link status changes. Node 1 decides that link A is down when it receives a message on link B. When link B fails, node 1 decides that link A is back up. Then, when a messages finally arrives from node 3, node 1 again decides that link A is down. Notice that node 1 sends three messages to node 3 concerning link A, whereas A's status has actually changed only once.

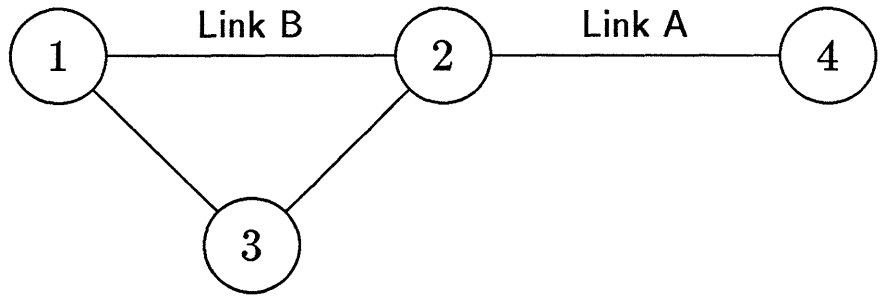

Figure 2a. **An Example Network.** All links are initially up. Link A fails. Shortly afterward, link B fails.

Event	Description
1	Link A fails. (2→1, $A\downarrow$) and (2→3, $A\downarrow$) sent and received.
2	(3→2, $A\downarrow$), (1→3, $A\downarrow$), and (1→2, $A\downarrow$) sent and received. (3→1, $A\downarrow$) sent but **not** yet received.
3	Link B fails. (1→3, $B\downarrow$), (1→3, $A\uparrow$), and (2→3, $B\downarrow$) sent and received.
4	(3→1, $A\downarrow$) that was sent in Event 2 arrives. (1→3, $A\downarrow$) sent and received.

Figure 2b. **Sequence of Events.** The table shows the messages sent in response to the change in status of links A and B. *Notation:* "($i\rightarrow j$, $l\downarrow$)" indicates a message sent from node i to node j saying that link l is down.

Event	T^1	T^1_2	T^1_3	T^2	T^2_1	T^2_3	T^2_4	T^3	T^3_1	T^3_2	T^4	T^4_2
1	du	du	uu	du	du	du	du	du	uu	du	du	du
2	du	du	uu	du	du	du	du	du	du	du	du	du
3	ud	dd	ud	dd	dd	dd	dd	dd	ud	dd	du	du
4	dd	dd	dd	dd	dd	dd	dd	dd	dd	dd	du	du

Figure 2c. **Topology Table Contents.** All topology table updating is assumed to take place immediately following an event. *Notation:* "du" means that in the indicated topology table, the entry for link A is down and the entry for link B is up.

Figure 2: SPTA Operation for Multiple Link Failures.

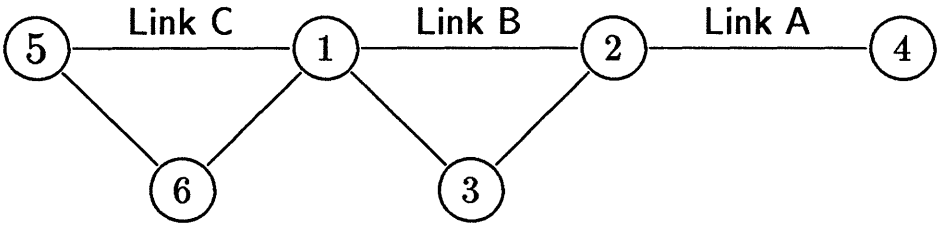

Figure 3: An extension of the network in Figure 2a.

Now consider extending the network of Figure 2a by adding two nodes to the left of node 1. The resulting network is shown in Figure 3, and involves three links which change status. Let links A and B fail as before. Node again 1 experiences three status changes concerning link A. It sends three messages to node 5 along link C, and along the path through node 6. Let the messages sent over link C arrive at node 5 before those sent via node 6. When they are received, node 5 has three status changes concerning link A. Now let link C fail. Node 5 examines its port topology table associated with node 6 and decides that A is up. Now let the three messages sent above by node 1 arrive at node 5 via node 6. This results in three more topology changes at node 5 concerning link A, for a total of seven!

Consider continually extending the network of Figure 3 as before, such that each extension adds another link which changes status. If there are k links which change status, then the leftmost node can have $2^k - 1$ status changes concerning the rightmost link. Since a node sends a message each time its main topology table changes, the number of messages sent in this example increases exponentially with the number of links which change status.

Acknowledgement

The authors thank Dr. Dimitri P. Bertsekas for his help in improving the presentation of SPTA.

References

[1] B. Awerbuch, "Locality-Based Adaptation of Communication Protocols to Dynamic Input and Network Topology," MIT Laboratory for Computer Science, April, 1987.

[2] A. E. Baratz and A. Segall, "Reliable Link Initialization Procedures," IBM Thomas J. Watson Research Center report, RC 10032, Revised 8/5/83.

[3] E. W. Dijkstra, "A Note on Two Problems in Connection With Graphs," *Numer. Math.* , Vol. 1, pp. 269–271, 1959.

[4] S. G. Finn, "Resynch Procedures and a Fail-Safe Network Protocol," *IEEE Trans. Comm.,* Vol. COM–27, No. 6, June 1979.

[5] E. Gafni "Topology Resynchronization: A New Paradigm for Fault Tolerance in Distributed Algorithms," Computer Science Dept., UCLA, 1987.

[6] R. G. Gallager, "Distributed Minimum Hop Algorithms," M.I.T. Laboratory for Information and Decision Systems Report LIDS-P-1175, Jan. 1982.

[7] J. M. Jaffe and F. H. Moss, "A Responsive Distributed Routing Algorithm for Computer Networks," *IEEE Trans. Comm.,* Vol. COM–30, No. 7, July 1982.

[8] J. M. McQuillan, I. Richer, and E. C. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Trans. Comm.*, Vol. COM–28, No. 5, May 1980.

[9] R. Perlman, "Fault-Tolerant Broadcast of Routing Information," *Computer Networks,* Vol. 7, Dec. 1983.

[10] E. C. Rosen, "Vulnerabilities of Network Control Protocols: An Example," *Computer Communications Review*, July 1981.

[11] J. A. Roskind, "Edge Display Spanning Trees and Resynchronization in Data Communication Networks," Ph.D. Thesis, M.I.T., and M.I.T. Laboratory for Information and Decision Systems Report LIDS-TH-1332, Oct. 1983.

[12] A. Segall, "Distributed Network Protocols," *IEEE Trans. Inform. Theory,* Vol. IT–29, No. 1, Jan. 1983.

[13] A. Segall and B. Awerbuch, "A Reliable Broadcast Protocol," *IEEE Trans. Comm.,* Vol. COM–31, No. 7, July 1983.

[14] S. R. Soloway and P. A. Humblet, "On Distributed Network Protocols for Changing Topologies," M.I.T. Laboratory for Information and Decision Systems Report LIDS-P-1564, May 1986.

[15] J. M. Spinelli, "Broadcasting Topology and Routing Information in Computer Networks," S.M. Thesis, M.I.T., and M.I.T. Laboratory for Information and Decision Systems Report LIDS-TH-1470, May 1985.

[16] U. Vishkin, "An Efficient Distributed Orientation Algorithm," *IEEE Trans. Inform. Theory,* Vol. IT-29, No. 4, July 1983.