A Protocol to Maintain a Minimum Spanning Tree in a Dynamic Topology *

C. Cheng, I.A. Cimet†, and Srikanta P.R. Kumar

Electrical Engineering and Computer Science Department
Northwestern University
Evanston, IL 60208

Abstract

We present a distributed protocol for updating and maintaining a minimum-weight spanning tree (MST) in a network with changing topology. The protocol can respond to multiple link/node failures and recoveries that can occur at arbitrary times. Given that an arbitrary finite number of topological changes occur during a period, the protocol finds the MST corresponding to the latest network, within finite time after the last change. The message complexity of the protocol is O(m|E|+k|V|) when k link recoveries and m link failures occur, where |V| and |E| are the total number of nodes and links, respectively.

1. Introduction.

A spanning tree of a communication network represents a minimally connected structure containing all the nodes in the network. Such trees are employed in point-to-point networks for efficient message broadcasting [8], in selecting a 'leader' processor [10], or to ascertain certain conditions (e.g.: mutual exclusion or termination) [9]. Spanning trees have also been used in local-area networks for gateway routing [12] and in multihop radio networks for bandwidth allocation [5].

In general, a cost or weight (representing, for example, usage fees or delay) is associated with the links of the network. Therefore, the spanning tree whose edges have minimum weight is usually needed in the tasks mentioned above. This tree is called the Minimum-Weight Spanning Tree (MST). One of the first MST protocols for point-to-point networks (hereafter denoted as the GHS protocol) was presented by Gallager, Humblet, and Spira in [11]. In a network of n nodes and e edges, the GHS protocol transmits O(nlogn+e) messages, with O(logn) bits/message, and uses O(nlogn) time units. (The time complexity of this protocol has been recently improved [2, 4, 10]).

For the most part, protocols run over a communication network with changing topology. The changes in topology entail additions and deletions of nodes and links over the course of time. Hence, many protocols that are able to run in spite of topological changes have been designed in the past decade (e.g.: [1, 6, 13]). These protocols are called *resilient* or *reliable*. In particular, a resilient protocol for the MST problem was proposed in [7].

In this paper, we present a new resilient distributed MST protocol. Given an initial MST, the protocol responds to topological changes caused by link/node failures and recoveries, and updates the MST. Multiple failures/recoveries can occur simultaneously or can overlap in time (i.e.: a failure or recovery can occur while another is being processed). Assuming that a finite number of topological changes occur at arbitrary times during a period (and no more changes after this period), it will be shown that the protocol will obtain the MST corresponding to the latest network within a finite time after the last topological change. Given m link failures and k link recoveries, the protocol uses O(kn+me) messages, each with O(logn) bits. In contrast, if one uses a protocol (such as GHS) to reconstruct the tree after every failure or recovery, the message complexity will be O((m+k)*(nlogn+e)), with O(logn) bits/message. The protocol presented herein is more efficient for dense networks than the resilient MST protocol in [7] which uses O((m+k)*n) messages with O(e) bits/message.

The paper is organized as follows. The distributed computation and communication model are given in Section 2. Section 3 explains the underlying graph theoretic properties and the main features of the distributed implementation. Section 4 describes the resilient

* This work was supported in part by grants from US-WEST Advanced Technologies, and from Bell Northern Research.

† I.A. Cimet was with the EECS Dept., Northwestern University. He is now with AT&T Bell Laboratories, Naperville, IL, 60566.

MST protocol. Section 5 presents a sketch of the proof of correctness. Finally, Section 6 contains the complexity analysis.

2. Computational Model.

The environment for the protocol is an asynchronous network represented by an undirected weighted graph G(V,E), where V is the set of nodes and $E \subset V \times V$ is the set of links. Each node, assigned with a unique identity number, is a computing unit involving a processor, a local memory, and also an input queue and an output queue with unlimited capacity. Each link (u,v), assigned with a weight $w(u,v)$, is a bidirectional communication line connecting nodes u and v. Each node knows only its local environment (the identity and the weights of its links) and follows the same protocol which consists of sending and receiving messages over the adjoining links, and processing these messages. The received (sent) messages are put in the input (output) queue on a first-in-first-out basis, and are also processed in that order.

A communication link in a dynamic network has the following properties. Messages can only be sent and received over a link which is functioning. However, a message need not arrive at the receiver, as the link may fail during transmission. When the link is functioning, messages can be independently transmitted in both directions, and they arrive at the other end after a finite undetermined delay, without error and in sequence. Whenever a link fails or recovers, both ends are notified in finite time but not necessarily at the same time. At the time when a link recovers, there are no messages in transit through it, nor are there messages waiting to be sent over it (i.e., all messages sent out for transmission on a link are deleted after the link fails). Observe that these properties are commonly used in the literature (see [1, 13]).

3. The Resilient MST Protocol.

Assume that a graph G(V,E) is given initially along with its MST. Furthermore, suppose that the weights of all the links are distinct (see [11]). Thereafter, a finite number of link failures and recoveries occur at arbitrary times. The resilient protocol must find, within a finite time after the last topological change, an MST corresponding to the latest graph.

Definition 3.1: Let $c_i = < e_i , t_i >$ be the i-th topological change in the network, where t_i is the time when the change occurs and e_i is the link that fails or recovers at that time. Let $E_{i+1} = E_i \cup \{e_i\}$ if c_i is a recovery, and $E_{i+1} = E_i - \{e_i\}$ if c_i is a failure, with $E_0 = E$. Similarly, define V_i to be the set of nodes in the graph after c_i. Then the *surviving network* after the i-th topological change is denoted by $S_i = G(V_i , E_i)$.

Suppose that only k topological changes occur in the network. Then, the distributed algorithm must find an MST for S_k.

3.1. Underlying Graph Theoretic Properties.

Before the distributed implementation is described, we outline what is involved in responding to link failures and recoveries, and the graph theoretic properties underlying the reconstruction of the MST. These properties will be the basis for the collective action of the nodes involved in a failure/recovery process. For convenience, the following notation will be adopted. Given a tree T, let T-e and T+e respectively denote the deletion and addition of edge e to T.

First, consider a single link failure. Let T be the MST of the graph G(V,E) prior to the failure, and let T' be the MST of G(V,E-{e}), where $e \in E$ is the link that fails. If e is a non-tree link, then T'=T and no updating of the MST is needed. Conversely, if e is a tree link, then this failure splits T into two fragments T_1 and T_2. These two fragments must be reconnected, if at all possible, to obtain the new MST. It can be observed that T'=T-e+e', where e' is the minimum weight link connecting a node in T_1 to a node in T_2 (see [7]). (If there is no link connecting T_1 and T_2, then the graph resulting from the failure is disconnected).

Now, when there are many, possibly simultaneous, failures (assume no recoveries, for the moment), the initial MST is broken down into many fragments. Each of these fragments is also a fragment of the MST of the final surviving graph. Hence, starting from these fragments, the MST for the final graph can be constructed by finding the links, connecting these fragments, whose weight is minimum. The distributed implementation of the response to link failures (described later) will be based on this idea, and being so, this part of the protocol has many similarities with the fixed-topology MST protocol of GHS [11]. However, there are some important differences, and these are explained in the next section.

To understand the response to a link recovery, consider a single recovery. If one of the end nodes of the recovered link is not in the original graph, then T'=T+e. (If both ends are not in the original graph, then there is no spanning for the graph resulting from this recovery). If both end nodes are in the original graph, then the recovered link forms a unique cycle with the existing tree links. The following proposition

indicates how the new MST can be obtained.

Proposition 3.1: Let T be the MST of G(V,E). Suppose link e=(u,v) recovers. If T' is the MST of G(V,E∪{e}) and u,v∈V, then T'=T+e-e' where e' is the link with the maximum weight among the edges that form the cycle in T+e. If one of u and v does not belong to V, then T'=T+e.

In order to extend the approach suggested by Proposition 3.1 to handle many simultaneous recoveries, it is necessary to sequentially apply the rule given by this Proposition. To see this, consider the recovery of links e_1 and e_2. The rule can be applied to both recoveries only if the cycles formed by $T+e_1$ and $T+e_2$ are disjoint. If the cycles overlap (as in Fig. 1), then the rule in Proposition 3.1 must be first applied to one of the recoveries and then the remaining recovery may be processed on the resulting graph. The order in which these recoveries are processed is not important (see [3]). Moreover, this rule is also applicable to recoveries within fragments. If a link recovers within a fragment, it forms a cycle with some of the tree links in the fragment. The new fragment derived according to the above rule is also a fragment (a subtree of the final MST).

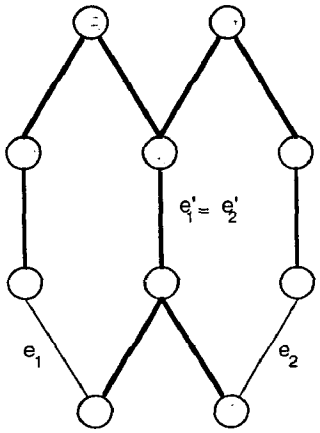

Fig. 1

The properties described above provide a basis for the resilient MST protocol, the main features of which are explained in the next section.

3.2. Main Features of The Distributed Implementation.

In a changing network environment, link failures and recoveries can occur simultaneously, or one event can occur when another event is being processed. At any moment, there can be several fragments that will have to be eventually combined. Recoveries can happen within or across fragments, and the appropriate cycle has to be identified to update the fragment and obtain the final MST. Each of the end nodes detecting the failure/recovery of a link initiates a response process, which will involve several other nodes in the task to be accomplished. Such response processes may overlap in the sense that a node may be involved in more than one of them at the same time. Before we present the description of the protocol, we briefly outline some of the main features of each of the response processes and their interaction.

3.2.1. Response to a Link Failure

The response to a link failure is based on the fragment expansion notion, and is similar in nature to the fixed topology distributed MST algorithm of the GHS protocol [11]. A failure response initiated by a node should first assign a unique fragment identity to the fragment to which the initiated node belongs. That is necessary to distinguish the ingoing and outgoing edges of the fragment, and find the minimum outgoing edge of the fragment (see [11]). In changing topology environments, the failures break the MST into several fragments. Several difficulties arise in attempting to modify the fixed-topology GHS algorithm to this context. The first issue concerns the fragment identity. When a tree link e=(u,v) fails within a fragment, two fragments are formed both of which must have separate identities. A natural thought is to assign the identities (w(e),u) and (w(e),v) to the fragments which contain node u and node v, respectively. As link weights and node identities are distinct, one might think this will suffice. However, this assignment may lead to two nodes in separate fragments having the same fragment identity, at the time when the edge connecting these nodes is tested (to see if it is outgoing), and thus, a correct MST may not result. In fact, it is not hard to see that the same undesirable situation results if the assignment of fragment identity is based on only a part of the fragment (i.e., node identities and link weights of any proper subset of the fragment).

There are two ways to avoid the situation described above, and maintain distinct identities for different fragments. The first approach is to let the fragment identity be the set of node identities of all the nodes in that fragment. With this assignment, two nodes in different fragments will have different identities when testing the edge connecting them. This approach has been used in a similar problem by Segall in [13]. However, with this approach the message size and the overall bit complexity will increase, as a substantial number of messages in the resilient MST protocol will contain the fragment identity.

An alternative approach is to have a counter for each link, which keeps track of how many times the link has failed and recovered, and attach this counter value to the fragment identity. Specially, suppose link e=(u,v) fails. Then, if k(e) is the counter value of this link immediately after its failure, assign the identity (w(e),u,k(e)) to the fragment containing node u, and the identity (w(e),v,k(e)) to the fragment containing node v. This would ensure that a new and distinct identity is generated for a fragment every time a link fails. Unfortunately, the counter cannot be bounded without some assumptions on the time or the order of the topological changes. However this approach may still be viable in practice for reasonably large counters that are good enough for long operating intervals. Notice that the size of the messages containing the fragment identity will be |V| times larger in the first approach than with the second.

The resilient MST protocol described in the next section could be used with either one of the approaches described above for assigning fragment identities.

3.2.2. Response to Link Recovery.

When a link e=(u,v) recovers, then both end nodes u and v exchange their fragment identities. If both are in the same fragment, then each node initiates a process to find the maximum weight edge in the cycle by propagating a message upward (note that fragments will be directed trees) all the way to the root of the fragment. The messages from u and v will 'meet' at a node, which is denoted as the least common ancestor (lca) of u and v. In other words, lca(u,v) will be the node which receives messages corresponding to the same recovery, from two of its descendant (or child) nodes. This node, which will have the information about the maximum weight in the cycle, takes action to delete and insert the appropriate links into the MST. If the cycle is intact (i.e., links in the cycle do not fail) throughout the recovery process, the goal of this process will be accomplished. Observe that when several recoveries occur simultaneously in the same fragment and the corresponding cycles are disjoint, then each recovery process can be performed independently of the others. However, if the cycles overlap, their processing is sequentialized in the following manner. When a node is involved in processing one recovery, it does not forward messages received (from its descendants) about other recoveries. Such a node stores the messages until the first recovery process is completed at which time one of the stopped messages will be forwarded upwards.

As messages of one recovery, on their journey towards the root, could get stopped by other recovery processes, a potential deadlock situation may occur. (Referring to Fig 2, both lca(e) and lca(e') will not receive messages on one side of the cycle). As each node involved in recovery processing will forward at least one recovery message, the root of the fragment will be informed of at least one recovery. The root will select the first arriving recovery, for processing, by sending a privilege message which travels down over the same path on which recovery message arrived and around the cycle to release the stopped messages. (In Fig. 2, the privilege will travel from the root to u, and then to v and upwards releasing the stopped message at x). After the cycle is updated, the root is informed of the completion. Thus, the root sequentializes the recovery processing also avoiding any deadlock. More implementation details are given in section 4.

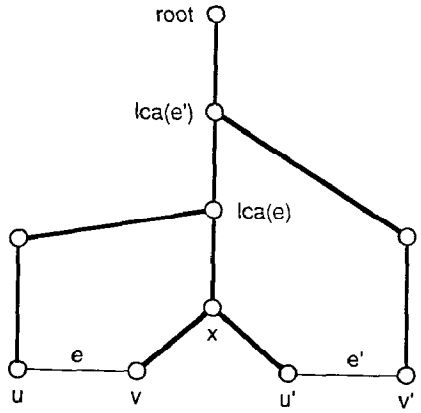

Fig. 2

3.2.3. Interaction of Failure and Recovery Process.

A fragment F can be involved in finding its minimum outgoing edge and concurrently process one or more link recoveries within it, as described earlier. As long as no tree link in F fails, all the processes will complete and update the fragment correctly.

A note of caution is in order. A node should not consider any of its recovered links as a candidate for its minimum outgoing edge when it is engaged in testing its outgoing edges, as this may result in the MST being incorrect. (For example in Fig. 3, links e_1 and e_3 may be included in the MST if recovered links are not considered, whereas e_2 should be included instead of e_3.)

When tree links in F fail, then each new fragment generated can again process the failures and recoveries within it. If a tree link failure in F causes the nodes u and v of a link recovery e=(u,v) to be in two different fragments, then the recovery process of e will stop until

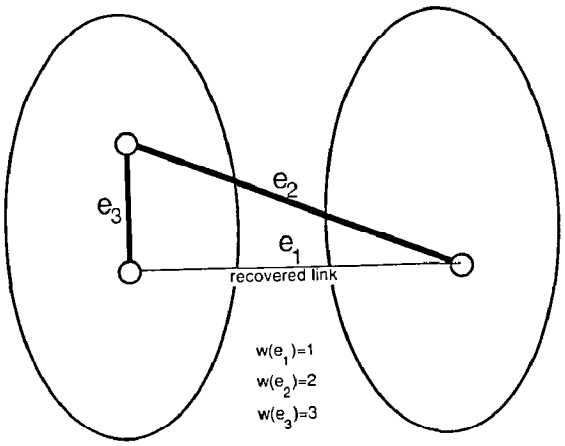

Fig. 3

the fragments merge and nodes u and v belong to the same fragment. Conversely, if the outgoing links of a fragment are all recovered links then only one of them will be selected for fragment expansion, as described in detail in the next section.

4. Description of the Resilient MST Protocol.

In this section, we describe the actions taken by a node involved in updating the MST, and the messages used. (A formal description is given in [3]). It is assumed that the initial MST is a directed rooted tree. Initially, all nodes are in *sleep* state. First, we shall describe separately the normal courses of failure and recovery processes. The interaction when failures and recoveries overlap in time is described later.

4.1. Failure Response.

When a node detects the failure of any its adjacent *non-tree links or unprocessed recovered links*, it simply deletes it from its local memory.

Consider any tree link e=(u,v), where node u is the parent of node v. If this link fails, node u (in *sleep* state) sends a message *FAILURE*<id(e)> to its parent, where id(e) is the new identity generated by the failure of e (as explained in section 3.2.1). This message will be propagated upwards to the root. The root upon receiving this message enters into a *reiden* state, and changes its identity to be id(e). Node v, on the other hand, marks itself as the root (of the fragment containing v), and also enters the *reiden* state.

Upon entering the *reiden* state, the root enters a broadcast-echo phase in which it broadcasts the message *REIDEN*<id> which travels down the tree to all nodes in the fragment. Every node that receives such a message, changes its state to *reiden* and its fragment identity to the identity contained in the message. Each node responds with a *REIDEN_ACK*<id> message which will be sent upwards from the leaves of the fragment to the root. When this message is received by the root then all nodes know the current fragment identity.

To find the minimum outgoing edge, the root enters the *find* state and performs another broadcast-with-echo, in which the root broadcasts a *FINDMOE* message over the fragment. A node receiving this message also enters into the *find* state and sends a *TEST*<id> message over each of its non-tree links. A node that receives such a message may respond with an *ACCEPT*<id> or with a *REJECT*<id>, which indicates whether or not the non-tree link is outgoing. (Note that the id sent with the *TEST* message must be returned in the response, as the id may change in the mean time due to other failures. The reason is the same for attaching the id to the echo in every broadcast-echo phase). During the echo, node u sends to its parent a *FINDMOE_ACK*<id,w_u> message , where w_u is the minimum weight of the outgoing edge found by the descendants of u. After a node sends this message it enters the *found* state. When the root receives a *FINDMOE_ACK* message from all its children it knows the weight of the minimum outgoing edge (or *moe*) of the fragment. At this point the root sends a *CHANGE_ROOT*<id> over the path that leads to the *moe*. If no other topological changes occur, the resilient protocol then proceeds in a manner similar to the GHS protocol, ie., when the end node of the *moe* receives the *CHANGE_ROOT*<id> it sends a *CONNECT* message to over the outgoing edge to merge the two fragments and starts a process over the "extended" fragment to find a new *moe*. Note that if there are m fragments, each finding its own *moe*, then at least two of them will choose the same outgoing edge.

Suppose that other topological changes occur while this process is being performed. For instance, assume that link e'=(u',v') fails and u' is the parent of v' in the fragment. Such a failure divides the fragment into two more fragments. As explained before v' will become the root of one of the fragments. However, u' will send a *FAILURE*<id'> message to the root of its fragment, where id' is the new identity corresponding to the most recent failure. When the root receives such a message it will start a new failure process corresponding to the last failure by giving a new id to its current fragment and finding a new outgoing edge. Observe that the failure process initiated by the failure of e will be abandoned because the original fragment created by that failure has now been divided. A node will ignore any messages that contain the identity given to the original fragment.

If the root of a fragment finds that there is no outgoing edge (i.e., either the fragment spans the entire network or it is disconnected from the rest of the network), then it broadcasts a *GOSLEEP* message over the and enters into the *sleep* state. A node that receives such a message will also into the *sleep* state.

Based on the above description of the protocol implementation for link failures, the following proposition is presented (the proof can be found in [3]).

Proposition 4.1: Corresponding to a set of failures (suppose that there is no recovery), if the final graph is still a connected graph, then the tree maintained by the protocol is the MST of the final graph and each node returns to the *sleep* state; otherwise, if the final graph becomes disconnected, then, in each connected component, the protocol maintains the MST of this subgraph and each node in each connected component will enter the *sleep* state.

4.2. Recovery Response.

Consider a recovered link e=(u,v). Initially all nodes are in a *normal* state (with respect to recoveries). The end nodes that detect the recovery, exchange *ID_CHECK*<id> messages where id is the fragment identity at each node, change their states to *recover*, and send a *RECOVERY*<id,w(e),w(p)>, where w(p) is the weight of the link going to the parent. A node that receives a *RECOVERY*<id,w(e),w> checks if the fragment id is the same. If the fragment id is not the same then a failure has happened and the fragment is being reidentified. Thus the message is simply ignored. Conversely, if the id is the same then the node changes its state to *recover* and sends a *RECOVERY*<id,w(e),w'> to its parents, where w'=max{w,w(p)} where w(p) is again the weight of the link going to the parent of the node. This process is continued until the message arrives at the *least common ancestor* of u and v, denoted by *lca(u,v)*. This node will receive a *RECOVERY* message from u and from v. Suppose that the message from v arrives first. Then the *RECOVERY* message is forwarded to the parent of the *lca* node as explained above. Conversely, the message from u will not be forwarded. Instead it will be stopped at the *lca* or further action. When the root receives the *RECOVERY*<id,w,w'> message from a child it responds with a *PRIVILEGE*<id,w,0> message which functions as a token that permits the serialization of the recoveries (see Section 3.2.2). If this message reaches the *lca(u,v)* after the second *RECOVERY* from v then this *PRIVILEGE* is returned to the root because no serialization is necessary. Conversely, if the second *RECOVERY* message has not been received then the *PRIVILEGE* message is forwarded over the cycle. If the *PRIVILEGE* reaches a node that has stopped the *RECOVERY* message (i.e. because another recovery is taking place) then the node will release the message and the *RECOVERY* will reach the *lca(u,v)*. At this point the *PRIVILEGE* is returned to the root. If there is another recovery, the root will send a new *PRIVILEGE* over the appropriate path.

When the *lca(u,v)* finally receives the two *RECOVERY* messages it starts the process that will replace the edge in the cycle with the maximum weight. To do this, it sends a *REPLACE*<id,w(e),w> over the cycle. If the node receiving the message is adjacent to the link with the maximum weight then it marks the link as a non-tree link, changes its state to *normal*, and forwards a *REPLACE*<id,w(e),∞>. On the other hand, a node that is not adjacent to the link with the maximum weight will change its state to *normal* and forward the original *REPLACE* message. A node that receives a *REPLACE*<id,w(e),∞> will reverse the parent-child relation.

It should be clear that in the implementation described above, the recovery processes proceed sequentially as suggested in Section 3.2.2.

Based on the above description of the protocol implementation for the link recoveries, the following proposition is presented (the proof can be found in [3]).

Proposition 4.2: Corresponding to a set of recoveries (suppose that there are no failures), if the final graph is still a connected graph, then the tree maintained by the protocol is the MST of the final graph and each node returns to *sleep* state; otherwise, if the final graph becomes disconnected, then, in each connected component, the protocol still maintains the MST of this subgraph and each node in each connected component also enters *sleep* state.

4.3. Interaction of Failures and Recoveries.

Suppose that a node u is the *lca* of an ongoing recovery process. Furthermore, assume that u is also participating in a failure process. Then u will hold the *REPLACE* message until the failure process is finished. Note that this implies that a failure process may cause a delay in the recovery process. However, this only happens in some instances.

Suppose now that one of the link in the cycle of an ongoing recovery process fails. Then the nodes in the fragment will receive a new identity and the messages generated by the recovery will be discarded because they will contain the old identity. The end nodes of the recovered link will start a new recovery process (after they receive their new identities). Note that the end nodes can "remember" the recovery because the link

remains unclassified.

We have described how a failure may affect a recovery process. However, a link recovery may also affect a failure process as follows.

Assume that an *lca* node has sent a *REPLACE* message and is waiting for its return. At this time the node will hold any message that will travel over the cycle because the topology of the cycle is being adjusted. These messages will be sent when the *REPLACE* message is returned. Similarly, a node that is not an *lca* will hold any *FAILURE* message when it is involved in a recovery process because the topology of the cycle is being changed and the parent-child relations are being changed.

It can be observed that Propositions 4.1 and 4.2 still hold when failures and recoveries occur simultaneously (see [3]). [1]

5. Proof of Correctness.

In the following, we briefly outline the correctness proof of the protocol.

Let G_P^t denote the physical network at time t i.e., the nodes and the links that are operational at time t. Define G_L^t to be the logical network corresponding to the information stored at each node at time t. In other words, a link (x,y) is in G_L^t if both node x and node y have recorded (x,y) as operational. (Note that, due to the delays in the notifications, G_P^t and G_L^t may not be identical at all times). It is also convenient to define a computational network, denoted as G_C^t, which is the logical network without the recovered links i.e., $V(G_C^t) = V(G_L^t)$ and $E(G_C^t) = E(G_L^t) - R^t$, where an edge is in R^t if either or both of its end nodes have recorded it as a recovered edge.

Property 5.1: At any time t during the computation, if a link (u,v) is classified as a tree link by node u, then (u,v) is in the MST of the component of G_C^t which includes (u,v).

Proof: Suppose that link (u,v) was a non-tree link before time t and becomes a tree link at time t. Then, due to Proposition 4.1 which holds even when there is mixture of failure processes and recovery processes as described in section 4.3, (u,v) must be an *moe* of some specific set of nodes defined on G_C^t. Therefore, based on the fragment expansion property, (u,v) is in the MST of the component, defined on G_C^t, which includes (u,v).

[1] Also see [3] for a further discussion on the interactions between failure and recovery processes.

Suppose that link (u,v) was a recovered link before time t and becomes a tree link at time t. Then, either (u,v) is inside some fragment and there is a tree link (x,y), in the cycle formed by adding (u,v) into this fragment, whose weight is greater than the weight of (u,v), and (x,y) has been deleted before the installation of (u,v) as a tree link; or there are only recovered links connecting the component, including node u, to other components of G_C^t. In the first case, due to Proposition 4.2 which holds when there are simultaneous recovery and failure processes, (u,v) is with the weight less than any outgoing non-tree link of the induced subgraph corresponding to the fragment rooted by y, assuming that x is the parent of y. Thus, based on Proposition 3.1 (u,v) is in the MST of G_C^t which includes (u,v) at time t. In the second case, (u,v) will be the only recovered link becoming a tree link at time t. Thus, (u,v) is the *moe* of the of the induced subgraph corresponding to the fragment including node u. Therefore, in either case, (u,v) is in the MST of the component, defined on G_C^t, which includes (u,v). □

Now, the correctness of the protocol can be established.

Theorem 5.1: The algorithm constructs an MST for each component of the final graph.

Proof: Since the notification of topological change on any link will arrive at the two end nodes in finite time, there exists t_1 s.t. $G_L^t = G_P^t$ for all $t > t_1$. Besides, the failure process will terminate at time t only if it finds the *moe* which is not selected by any other failure process and the connection is successfully done at time t, or the fragment where this failure process originated becomes disconnected with all other fragments in G_C^t. Also, in the case that there are only recovered links as the outgoing edges of some fragment, then one and only one of them, say (u,v), will be used to merge the two fragments including u and v respectively. Thus, in finite time after the last topological change there will be one spanning tree on every component of the final graph. Therefore, from that time onward, every recovered link will be inside some fragment. This implies that each recovered link will either become a tree link or a non-tree link in finite time since there are no deadlocks (as described in Section 4. Hence, there is a time t_2 s.t. $G_C^t = G_L^t$ for time $t > t_2$. If we let t_3 to be the maximum of t_1 and t_2, then $G_C^t = G_P^t$ for $t > t_3$. This fact, combined with Property 5.1 which holds throughout the computation, will imply that the spanning tree T on each component of G_C^t for $t > t_3$ is an MST of that component of the final graph. □

6. Complexity Analysis.

Suppose that the network has n nodes and e edges. Moreover, assume that a single failure occurs. It can be noted that the *FAILURE* message travels over the edges of a tree with O(n) messages. Similarly, broadcasting the new identity of the fragment, finding the new *moe*, and merging the fragments require O(n) messages since these activities are done over the edges of the tree. However, to find the minimum outgoing edge of the fragments, messages need to be sent over O(e) non-tree links. Thus the message complexity for a single failure is O(e) messages.

Suppose now that a single recovery occurs. As in the above case, the messages travel only over links of the tree and no non-tree links are traversed. Therefore, the complexity for a single recovery is O(n) messages.

As mentioned in previous Sections, some of the activity of a failure or a recovery may be lost because of subsequent failures (see Sections 4.1 and 4.3). Nevertheless, when a link failure occurs within a fragment F, it suspends any failure process in progress in F as well as the concurrent recovery processes in F (if any) and a completely new process is started. Note that for each of the suspended failure processes, the effort devoted to find its *moe* is lost. Therefore, the messages lost are bounded by 2e+6n. Similarly, suppose that all suspended recovery processes are taking place in disjoint cycles of the fragment. Then the messages in each process only traverse on the links of the relevant cycle. Thus, the total number of wasted messages is at most 3n. Therefore, the protocol uses only O(me+kn) messages to reconstruct an MST for the final graph where m and k are the numbers of tree link failures and recoveries respectively. Note that e and n refer to the maximum number of edges and nodes in the graph during the computation.

The best case message complexity for recoveries is O(n) when the cycles formed by these recoveries are disjoint. The best case message complexity for failures is O(e) when all the fragments except one choose the different links as their respective *moe*.

The worst case time complexity (evaluated by assuming that it takes one time unit to send a message across any link) is also equal to number of messages used during the computation. The best case time complexity is equal to O(H) where H is the maximum height among the fragments created during the computation.

References

1. B. Awerbuch and S. Even, "Reliable Broadcast Protocols in Unreliable Networks," *Networks*, vol. 16, no. 4, pp. 381-396, Dec. 1986.
2. B. Awerbuch, "Optimal Distributed Algorithm for Minimum Weight Spanning Tree, Counting, Leader Election, and Related Problems," *Symp. Theory of Comp.*, pp. 230-240, May 1987.
3. C. Cheng, "A Protocol to Maintain a Minimum Spanning Tree in a Dynamic Topology," M.S. Thesis, Northwestern University, Evanston, IL, June 1988.
4. F. Chin and H.F. Ting, "An Almost Linear Time and O(nlogn+e) Messages Distributed Algorithm for Minimum-Weight Spanning Trees," *26th Symp. on Foundations of Comp. Sci.*, pp. 257-266, Portland, OR, Oct. 1985.
5. I. Chlamtac and S. Kutten, "Tree-Based Broadcasting in Multihop Radio Networks," *IEEE Trans. on Comp.*, vol. C-36, no. 10, pp. 1209-1223, Oct. 1987.
6. I.A. Cimet and S.P.R. Kumar, "A Resilient Distributed Protocol for Network Synchronization," *ACM SIGCOMM Symp. Commun. Arch. and Protocols*, pp. 358-367, Stowe, VT, Aug. 1986. Also in *Computer Communications Review*, vol. 16, no. 3.
7. I.A. Cimet and S.P.R. Kumar, "A Resilient Distributed Algorithm for Minimum-Weight Spanning Trees," *Int'l Conf. on Parallel Proc.*, pp. 196-203, St. Charles, IL, Aug. 1987.
8. Y.K. Dalal and R.M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets," *Commun. ACM*, vol. 21, no. 12, pp. 1040-1048, Dec. 1978.
9. N. Francez, "Distributed Termination," *ACM TOPLAS*, vol. 2, no. 1, pp. 42-55, 1980.
10. E. Gafni, "Improvements in the Time Complexity of Two Message-Optimal Election Algorithms," *4th ACM Symp. on Principles of Dist. Comp.*, pp. 175-185, Minaki, Ont, Canada, Aug. 1985.
11. R.G. Gallager, P.A. Humblet, and P.M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Trans. Program. Lang. Syst.*, vol. 5, pp. 66-77, Jan. 1983.
12. R. Perlman, "Gateway routing," *safasasf*, pp. 111-111, asfasfsaf, date.
13. A. Segall, "Distributed Network Protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 1, pp. 23-35, Jan. 1983.

APPENDIX: Formal Description of the Resilient Algorithm

This appendix presents a brief formal description of the resilient protocol (for a complete description see [3]). It is assumed that the reader is familiar with the messages and variables used in the GHS algorithm. Also note that some trivial parts are omitted.

```
(1) Response of node u to failure notification for link (u,v).
if (u,v) is a tree link then
    generate new fragment identity fid.
    if u is a child of v then
        u becomes the root of fragment.
        f-state←reiden.
        id←fid.
        send REIDEN<id> to all children.
    else
        if f-state=found then
            hold FAILURE message.
        else
            if f-state=reiden or f-state=findmoe then
                send FAILURE<fid> to parent.
            end if
        end if
    end if

(2) Response of u to FAILURE<fid> over (u,v).
if u is root then
    id←fid;
    send REIDEN<id> to all children.
else
    send FAILURE<fid> to parent.
end if

(3) Response of u to REIDEN<fid> over (u,v).
id←fid.
f-state←reiden.
if u has no children then
    send REIDEN_ACK<id> to parent.
else
    send REIDEN<id> to all children.
end if

(4) Response of u to REIDEN_ACK<fid> over (u,v).
if id=fid then
    if u received REIDEN_ACK<fid> from every child ther
        if u is leader then
            send FINDMOE<id> to all children.
            f-state←find.
        else
            send REIDEN_ACK<id> to parent.
        end if
    end if
end if

(5) Response of u to FINDMOE<fid> over (u,v).
if id=fid then
    use TEST<id> to find all incident outgoing edges.
    if u has no children then
        send FINDMOE_ACK<id,moe> to parent.
        (moe is the local moe of u)
    else
        send FINDMOE<id> to all children.
    end if
end when

(6) Response of u to FINDMOE_ACK<fid,e> over (u,v).
if id=fid then
    if u is leader then
        send CHANGE_ROOT to child who reported moe.
    else
        if u received FINDMOE_ACK from
          every child then
            send FINDMOE_ACK<id,moe> to parent
                with local moe.
    end if
end if

(7) Response of u to CHANGE_ROOT<fid,e> over (u,v).
let a child of u be the parent.
if e is an incident link then
    send CONNECT over moe.
else
    send CHANGE_ROOT to child that reported e
      in FINDMOE_ACK.
end if

(8) Response of u to recovery notification for (u,v).
if u and v are in the same fragment then
    send RECOVERY to parent.
end if

(9) Response of u to RECOVERY<fid,w,w'> over (u,v).
if u received a RECOVERY message from node x
  for the same recovery then
    wait until u has received echo then
        send REPLACE to child reporting maximum weight
    end wait
end if
```