A Simple Distributed Algorithm for the Maintenance of a Spanning Tree

Brahim Hamid, Bertrand Le Saëc and Mohamed Mosbah
LaBRI-University of Bordeaux-1
351, cours de la libération
Talence, 33405, France
{hamid,lesaec,mosbah}@labri.fr

Abstract

This work is devoted to the problem of spanning tree maintenance in the presence of crash failures in a distributed environment using only local knowledge. Using a pre-constructed spanning tree of a $k$-connected graph, we present a protocol to maintain a spanning tree in the presence of $k-1$ consecutive failures. The contribution of this paper is twofold. First, the problem is formalized as an occurrence of the Menger’s theorem in distributed setting. The second result shows an implementation of the protocol which is composed of a set of modules encoded in the asynchronous message passing model. After each failure occurrence, our algorithm maintains a spanning tree in $O(N)$ time using $O(M+N)$ messages and $O(\Delta)$ bits per node. Here $\Delta$ is the degree, $M$ the number of edges and $N$ the number of nodes of the graph to be maintained. Furthermore, the studied network is semi-anonymous: Only the root needs to be identified.

Keywords: Distributed computing, failure detectors, fault-tolerance, local knowledge, maintenance, networks, spanning tree, vertex connectivity.

1. INTRODUCTION

Many applications in distributed environments are based on the network topology or structure knowledge. The designed protocols often used this knowledge as input information. In our case, the knowledge is the vertex connectivity applied to design protocols on unreliable networks modeled by graphs. For routing applications [11], each node uses information about its neighborhood to update the routing tables when the topology changes. So, it is suitable to maintain dedicated structures to guarantee the perennity of the routing. An important measure in this routing schemes is the number of nodes (or processors) to be updated upon a topology change. In our context, we use a spanning tree of the network described as a graph: A distributed system is represented as a connected, undirected graph denoted by $G = (V,E)$ where a node in $V$ represents a process and an edge in $E$ represents bidirectional communication link.

The Problem. Consider a system modeled by a graph $G = (V,E)$. Let $T = (V,E_T)$ be a spanning tree of $G$ rooted at a distinguished node $v_0$. We denote by $N$, where $N \geq 2$, the number of the nodes of $G$ and by $M$ its number of edges. The degree of $G$ is denoted by $\Delta$. It is known that for a graph with a distinguished node, a distributed computation of a spanning tree can be easily performed [6]. Therefore, the assumption of a pre-constructed spanning tree is not a loss of generality. The studied spanning tree maintenance problem is stated as follows. After the crash of a node $v$, we have to rebuild another spanning tree of $G$ deprived of $v$ with a minimum of changes. Obviously, if $G$ becomes disconnected this fails. It is assumed that a crash detection service is implemented using unreliable failure detectors [4, 5], satisfying: After the crash of some node, the crash detection service reaches a stable state and then all the neighbors are informed about the crash. The stability of the crash detection service is assumed to be reached in a finite time after the stability of its failure detectors. We consider a decentralized setting where nodes may crash by permanently halting and can perform only computations based on local knowledge.
Previous Works. The specific protocols devoted to maintain a spanning tree are proposed in [1, 3, 7, 9]. In [1], the authors studied the tree maintenance problem as the graph’s topological changes after edge failures. The maintained spanning tree is obtained using incremental update procedures, the time complexity of their solution is $O(\Delta \log M + N)$. In [3], an algorithm to maintain a common data is presented. It operates in a communication network where nodes are arranged in a chain is presented. The solution is an improvement of some previous solutions including those based on Full Broadcast and on Broadcast with Partial Knowledge [2]. The time complexity in both is $O(N + M)$. Therefore, the solution presented in [3] solved the Broadcast with Partial Knowledge problem in $O((M + N) \log M)$. The protocol presented in [7] is composed of iterations to rebuild a spanning tree of a maximal connected component when an edge fails. The time complexity of such a protocol is in $O(\text{“actual size”})$ of the biggest connected component adding an extra polylogarithmic time. Nevertheless these algorithms, either assume the network to be complete or the nodes have identities and deal with edge failures. Moreover, the notification service of failures are assumed to be reliable. In [9], the last assumption is not taken into account and node failures are considered. They, as described previously, use stable failure detectors. In order to encode such an algorithm, a structure called “expander graph” is implemented. This structure is more expensive than a simple spanning tree construction.

Our Contribution. In the presented work, we consider node failures in arbitrary network modeled by a graph. Here we assume that the graph $G$ is $k$-connected. This guarantees the success of the maintenance of a spanning tree in the presence of $k - 1$ consecutive crashes. Our protocol uses only local knowledge. That is, to perform a computing step, only information related to the states of the neighbors is required. Especially, each node knows its neighbors in $G$ and in the current spanning tree $T$ the “position” of a node is done by its father except for the root, and a set of ordered sons.

Our algorithm works without any more assumption for any node failure except the root. Moreover, this protocol does not need a completely identified network. Only the root needs to be identified: the network is semi-anonymous. To take into account the particular case of a root failure, we just need to have an extra information: A son of the root has been chosen to start the algorithm, in the sequel it will be called the “succesor son”. In the worst case, our algorithm takes $O(N)$ times and uses $O(M + N)$ messages and $O(\Delta)$ bits per node. Note that the complete used memory is $O(N \times \Delta)$ bits.

The rest of the paper is organized as follows. The model used to encode distributed algorithms is explained in Section 2. In Section 3, we present our approach, using formal modules, to deal with the problem of a maintenance of a spanning tree for 2-connected graphs in the presence of one failure. Section 4 shows the correctness of the presented protocol. Then, we extend the algorithm to treat $k - 1$ failures in $k$-connected graphs in Section 5. Analysis of the algorithm is given. Section 6 concludes the paper with a discussion about extended works.

2. PRELIMINARIES

The network is represented by an undirected graph in which each node represents a processor, and each edge is present between two nodes if their corresponding processors may “communicate”. An algorithm in such a system consists of a local program at each node. The program encodes the local actions that processor may make.

2.1. Graphs

$V$ is a set of nodes and $E$ a set of edges connecting nodes. A graph is a couple $(V, E)$ where $E \subseteq V^2$. We use sometimes the notations $V_G$, $E_G$ to denote respectively $V$, $E$ where $G = (V, E)$. Node $u$ is a neighbor of node $v$ if $(u, v) \in E$. A path $p$ in $G$ is a sequence $(v_0, \ldots, v_l)$ of nodes such that for each $i < l$, $(v_i, v_{i+1}) \in E$. The integer $l$ is called the length of $p$. A path $P = (v_0, \ldots, v_l)$ is simple if it does not contain cycles. The set of neighbors of node $u$ is denoted by $N(u) = \{v \in V / (u, v) \in E\}$. The degree of a node $v$, denoted $\text{deg}(v)$, is the number of neighbors
of \( v \). Then, \( \text{deg}(v) = \#\{ (v, u) \in E \text{ such that } u \in V \} \). The degree of \( G \) is \( \text{deg}(G) = \max\{\text{deg}(v) \text{ such that } v \in V\} \). A ball centered on \( u \) with radius \( l \) is the set \( B_l(u) = \{ u \} \cup \{ v_j \in V : \text{there exists a path } (v_0, ..., v_j) \text{ in } G \text{ with } v_0 = u \text{ and } j \leq l \} \). We will say that \( v \in V_G \) is a \( l \)--neighbor of \( u \) if \( v \) is in \( B_l(u) \).

We use \( d(u, v) \) to denote the length of the shortest path in \( G \) between \( u \) and \( v \). Then, the diameter of a graph \( G \), denoted by \( D(G) \), is the longest distance among all pairs of nodes in \( G \). Formally, \( D(G) = \max_{u,v \in V_G} d(u,v) \).

We say that the graph \( G' = (V', E') \) is a subgraph of the graph \( G = (V, E) \), if \( V' \subseteq V \) and \( E' \subseteq E \). A graph is connected if for any pair of its nodes, there exists a path between them. Let \( V' \subseteq V \), we denote by \( G|V' \) the restriction of \( G \) to the subgraph \((V', \{(u, v) \in E \cap V' \}). \) We also denote by \( G \setminus V' \) the graph \( G[(V \setminus V')]. \) When \( V' \) is a singleton \( V = \{ v \} \), by abuse, we will use the notations \( G|v \) and \( G \setminus v \).

A tree \( T = (V_T, E_T) \) is a connected graph \( G \) that has no cycle and an elected node \( v_0 \) called the root. As usual, we associate with a tree, a partial order \( \leq \) on \( V_T \) defined by \( u \leq v \) if there exists a simple path \( v_0, v_1, ... , v_i \) in \( T \) such that \( u = v_i \) and \( v = v_j \) with \( i \leq j \). If \( j = i + 1 \), \( u \) is the father of \( v \) and \( v \) is a son of \( u \). We denote by \( \text{Sons}(v) \) the set of the sons of \( v \). In the sequel, we assume that the list “Sons” of \( v \) is ordered. A node \( u \in V_T \) with no son is called a leaf of \( T \). Then, a spanning tree \( T = (V_T, E_T) \) of a graph \( G \) is a tree such that \( V_T = V_G \) and \( E_T \subseteq E_G \). The tree can be defined also by \( T(\text{Father}, \text{Sons}) \).

**Fact 2.1** Let \( T \) be a spanning tree of \( G \) with \( v_0 \) as root. Let \( v_0' \in V_T \) and \( p = (v_0, v_1, ... , v_l) \) the simple path in \( T \) where \( v_1 = v_0' \), then the tree \( T' \) with \( v_0' \) as root, associates with the partial order \( \preceq \) defined by :

- \( \forall v, v' \notin \{ v_0, v_1, ... , v_l \}, v \preceq v' \iff v \preceq v' \)
- \( \forall v, v' \in \{ v_0, v_1, ... , v_l \}, v \preceq v' \iff v \preceq v' \)

is also a spanning tree of \( G \).

We denote by \( T_u \) a tree with root \( v \). Formally, \( \forall u \in V_{T_u}, u \preceq v \). We denote by \( T(u) \setminus v \) the maximal subtree of \( T_u \) that contains the node \( u \), but not \( v \). A vertex \( v \) of a connected graph \( G \) is a “cut-node” iff \( G \setminus v \) is not connected.

**Definition 2.2** We denote by \( K_G \) the connectivity of a graph \( G = (V, E) \) which is the minimum size of any set \( V' = \{ v_1, ... , v_k \} \subseteq V \) such that \( G \setminus V' \) is not connected.

**Theorem 2.3** (Menger) A graph \( G = (V, E) \) is a \( k \)-connected graph if for any pair \( u, v \in V \) there exists \( k \) disjoint paths linking \( u \) and \( v \).

### 2.2. Message Passing Model

The model of the distributed system we will deal with is a point-to-point network of communicating entities. This system is modeled by a connected simple graph where each node represents an autonomous entity of calculation (e.g. thread, process, machine) and each edge a communication channel. The system is asynchronous; i.e. there is no global clock. The vertices have only local vision of the graph and communicate only with their neighbors by asynchronous messages. More precisely, a vertex \( v \) is equipped with ports, numbered from 0 to \((\text{deg}(v) - 1)\), which will be used to communicate with neighbors. The attribution of these numbers is completely arbitrary and does not depend on the identities of the neighboring nodes. For a sake of clarity, in the explanation of the algorithms we refer to a node instead of its corresponding port number. Note, however, that in our solution the identities of nodes are not required. We assume that, for a couple of neighboring vertices, the order of sending messages is the same as that of receiving them. For the algorithms presented in this paper, the network is anonymous which means that vertices have no identities.
A Simple Distributed Algorithm for the Maintenance of a Spanning Tree

Complexity measures in the message passing model are related to the number of messages and the amount of time as the local computations model. We will focus on the worst-case. So, we assume that each processor's state includes a subset of states to denote *terminated* states. After the reach of such states, actions of the program maps terminated states only to terminated states. Thus, we say that the algorithm has terminated (or ended) when all processors are in the terminated states and all messages are delivered. The message complexity of an algorithm implemented in the message passing model is the maximum of the total number of messages sent during all the possible executions of this algorithm. For the time complexity, we adopt the common approach assuming that the maximum message delay in any execution is one unit of time. Note that such a measure is not used to prove the correctness of such algorithms. Then, to calculate the time complexity of an algorithm it suffices to take the running time until the termination. A measure of a space complexity of a distributed algorithm is the size of the data stored at each node of the graph. Obviously, the concrete memory used is the size of all the data in the graph.

2.3. Spanning Tree Module

We present here the classical distributed algorithm to compute a spanning tree $T$ of a graph $G$, encoded in the message passing model. Since it will be used by our algorithm, it is presented as a "module" with parameters $^2$. This notion of module is similar to the interacting components used in [8]. The span polynomial module is referenced as follows:

$$STM(G; T; v_0; Stage; X; Y; Father; Sons)$$

where $G$ is the treated graph and the module builds a spanning tree $T$ rooted at $v_0$. The structure of $T$ is stored locally in each node using the labels $Father$ and $Sons$ described previously. Initially, all the nodes $v$ have their variables "Stage" valued to $X$. At the end of the computation, each node $v$ has its variable modified in such a way $Stage(v) = Y$. The following distributed algorithm describes an implementation of the spanning tree module.

Spanning Tree Module $STM(G, T; v_0; Stage, X, Y; Father; Sons)$

- **Input**: A graph $G = (V, E)$ and $v_0$ the chosen root.
  - **Variables**:
    - $B(v)$: the set of (locally) ordered immediate neighbors of $v$ which is an initial data.
    - $Stage(v)$: the state of $v$ can take many values. The only used in this module is: "X" as the initial value, "WA" to mean that $v$ has been included in the tree and "Y" to mean that $v$ finished locally its computation.
    - $Father(v)$ is the father of $v$ in the on-building spanning tree.
    - $Sons(v)$: are ordered list of the sons of $v$ in the previous tree.
    - $Included(v)$: the set of $v$'s neighbors included for the first time in the on-building spanning tree.
    - $Terminated(v)$: the set of $v$'s neighbors that finished locally the computation of the spanning tree.
    - $i, p, q$: integer;
  - **Initialization**:
    - $v_0 \in V$, $Stage(v_0) = X$.
- **Results**: A spanning tree $T = (V, E_T)$ of $G$ with root $v_0$, such that $\forall v \in V, Stage(v) = Y$.
- **Actions**:
  
  **ST-A1**: (For the initiator $v_0$ only, execute once:)
  1. $Stage(v_0) = WA$;
  2. $Father(v_0) := q$;
  3. $Sons(v_0) := \emptyset$;
  4. $Included(v_0) := \emptyset$;
  5. $Terminated(v_0) := \emptyset$;
  6. for $i := 1$ to $deg(v_0)$ do send $<st_init>$ via port $i$;

  **ST-A2**: (A message $<st_init>$ has arrived at $v$ from port $q$)
  
  1. If ($Stage(v) \neq WA$)
  2. $Stage(v) := WA$;
  3. $Father(v) := q$;
  4. $Sons(v) := \emptyset$;
  5. $Included(v) := \emptyset$;
  6. $Terminated(v) := \emptyset$;
  7. $send <st_init>$ via port $q$;
  8. If ($deg(v) = 1$)
  9. $Stage(v) := Y$;
  10. $send <st_back>$ via port $q$;
  11. else for $i := 1$ to $deg(v)$ do
  12. if ($i \neq q$)
  13. $send <st_init>$ via port $i$;
  14. else
  15. $Included(v) := Included(v) \cup \{q\}$;
  16. $Included(v) := Included(v) \cup \{q\}$;
  17. $Included(v) := Included(v) \cup \{q\}$;
  18. $send <st_back>$ via port $Father(v)$;

  **ST-A3**: (A message $<st_back>$ has arrived at $v$ from port $q$)

  1. $Sons(v) := Sons(v) \cup \{q\}$;
  2. $Included(v) := Included(v) \cup \{q\}$;
  3. $Included(v) := Included(v) \cup \{q\}$;
  4. $Included(v) := Included(v) \cup \{q\}$;
  5. $Included(v) := Included(v) \cup \{q\}$;
  6. $Included(v) := Included(v) \cup \{q\}$;
  7. $Included(v) := Included(v) \cup \{q\}$;
  8. $Included(v) := Included(v) \cup \{q\}$;
  9. $Included(v) := Included(v) \cup \{q\}$;
  10. $Included(v) := Included(v) \cup \{q\}$;
  11. $Included(v) := Included(v) \cup \{q\}$;

  **ST-A4**: (A message $<st_back>$ has arrived at $v$ from port $q$)

  1. if ($Included(v) \neq B(v)$ and $Terminated(v) \neq B(v)$)
  2. $Stage(v) := Y$;

$^2$The use of the term "module" is justified by the fact that the module behavior is not exactly those of classical "procedures" and/or "subroutines".
Here, the variable $Father$, set to $\bot$, means that the corresponding node has no defined father. At any step of the computation, when a node $v$, not yet in the tree ($\text{Stage}(v) \neq WA$) receives a $\text{st}_t\text{tok}$ message from its neighbor $w$, node $v$ includes itself in the tree by changing its $\text{Stage}$ to $WA$. Moreover, in the same time, $Father(v)$ is set to $w$, set “Included” is now composed of $w$ and set “Terminated” is initialized to $\emptyset$. Finally, $v$ informs $w$ to be add in its set of sons sending a $\text{st}_s\text{on}$ message. At the reception of such a message, $w$ adds $v$ in both its sets of sons and in the set of nodes that are included in the tree (“Included”). So at each execution of the rule $\text{STA}(1)$, the number of nodes not yet in the tree decreases by 1. When $v$ finds all its neighbors already included in the tree, it applies either the action $\text{STA}(2)$, $\text{STA}(6)$ or $\text{STA}(4)$. So it means that $v$ has locally terminated its computation, then it informs its father $\text{STA}(4)$. Note that this action is executed firstly by the leaves.

The computation finishes when all the nodes $v$ are such that $\text{Stage}(v) = Y$. And obviously we have a spanning tree of $G$ rooted at $v_0$ defined by the third components (or the fourth components) of the variables of the nodes. The root of the spanning tree is then the unique node with its father equals to $\bot$.

Then, the following property holds:

**Property 2.4** Given a graph $G = (V, E)$ and a chosen node $v_0$. The spanning tree module constructs a spanning tree of $G$ rooted at $v_0$ using $(2\#E + \#V - 1)$ messages.

In the sequel, we will need to define some modules to encode our algorithms. For a sake of uniformity, modules will use the following standard header format.

```
name (struct0, · · · struct4; node0, · · · node4; lab0, val0, val′0; · · · ; lab4, val4, val′4)
```

The header is composed of its name and a set of optional parameters. Each of the sets is separated using the character ". The first set of parameters is the structures of the manipulated graph, the second is a set of the distinguished nodes. The rest is related to used data (variables), their required initialization values and their expected values.

### 2.4. Crash Detection Module ($\text{CDM}(G; \text{Crashed})$).

The crash detection module is based on an unreliable failure detection service. Such a service may be implemented in asynchronous distributed systems using timeout and heartbeat strategy for example (see [4, 5]). The $\text{CDM}$ informs, only after the stability of its corresponding failure detector, each node about the possible crash of one of its neighbors. Thus, after such an indication, the failure detector of any non crashed node $v$ has detected all the crashed neighbors which are now stored in a list of crashed neighbors $\text{Crashed}(v)$. Moreover, all the sets used to encode the state of each process will be avoided from crashed processes. For example, the set $\text{Sons}(v)$ is composed of the sons of $v$ without crashed sons. If the father of a node $v$ is in $\text{Crashed}(v)$, then it is set to $\bot$. Thus, we denote by $\text{GST}$ the time after which the failure detectors of all nodes stabilize. We assume that the time unit used to measure the failure detector stability is the same as the one used to measure our distributed algorithm. Therefore, the cost of the $\text{CDM}$ is in $O(\text{GST})$ times and then uses also $O(\text{GST})$ messages for each failure occurrence.

### 3. PROTOCOL FOR 2-CONNECTED GRAPHS AND ONE FAILURE

In this section we present a protocol to maintain a spanning tree of 2-connected graphs in the presence of one failure. Our algorithm starts with a spanning tree of the graph $G$ associated with

$\text{GST}$ for Global Stabilization Time.
the network. Such a tree has been obtained using the same algorithm as the one given in the
previous section. We will use an occurrence of the well-known Menger’s theorem [12]:

Proposition 3.1 Let \( T \) be a spanning tree of a graph \( G \). Then the graph \( G \) is 2-connected iff
\( \forall v \in V_G, \exists E \subseteq E_G \setminus E_T \), such that \((V_G \setminus v,E_T \cup E)\) is a spanning tree of \((V_G \setminus v,E_G \setminus v)\) with
\( \#E \leq \#SON(v) \).

We consider a graph \( G \) with a spanning tree \( T \) rooted at \( v_0 \). When a node \( v_f \) crashes, using the
crash detection module, the neighbors of \( v_f \) know this crash. In this case when \( v_f \) is not the root
\( v_0 \), this information is propagated to \( v_0 \). Then, \( v_0 \) is in charge to rebuild a spanning tree of \( G \setminus v_f \).
In the particular case of a crash of \( v_0 \), a pre-chosen son of \( v_0 \) called a “sucor son” will assure
this charge. So during the construction of a spanning tree \( T \) of \( G \), the root \( v_0 \) chooses locally
one of its sons as its sucor son. We will use the label \( Suc \) to encode such an information. In \( T \), there is only one node with \( Suc \) equal to \( true \). In the sequel, the node in charge of the
construction of the new spanning tree will be called the “maintainer node”.

Now, we will first present an overview of our algorithm, referred to as \( MOST \) algorithm in the
rest of the paper. Then, we describe the three steps of the algorithm in more detail, proving
their correctness and analyzing their complexities using assumptions related to the asynchronous
message passing model. Let \( v_f \) be a crashed node.

1. detection of a crash. Neighbors of the crashed node detect that node \( v_f \) is crashed.
2. propagation. Father of the crashed node, if it exists, informs the root of the tree that some
node in its sons is crashed.
3. maintenance. Maintainer node updates the spanning tree of \( G \setminus v_f \).

Step two can be very short if it is the root that has crashed: The sucor son of the root becomes
the maintainer node. It is the only one \( Suc \) equal to \( true \). If \( v_f \) is not the root of \( T \), the father
of \( v_f \) generates an information to be propagated along the simple path linking it to \( v_0 \). Then
\( v_0 \) becomes the maintainer node. Such a task is done using a simple path propagator module
(see below). The third and last step is more delicate. It corresponds to the maintenance module
detailed in the sequel.

Simple Path Propagator Module \( SPPM(T;v;Traversed,X,Y) \). The aim of this module is to
propagate an information along a simple path linking node \( v \) to the root of a tree \( T = (V,E_T) \). We
will introduce the label \( Traversed \) to notice the expected distinguished simple path. Initially, all
the nodes have their variables “Traversed” valued \( X \). At the end of the application of this module
each node \( u \) in the simple path has its variable “Traversed” set to \( Y \). This module is implemented
using the first part of the root changing module. Thus, the application of the \( SPPM \) to \( T \) requires,
in the worst case, the use of \( \#V - 1 \) messages.

Maintenance Module \( MSTM(G,T;r) \). This module is applied by the maintainer node \( r \). It
works as follows. We denote by \( T' \) the “on-building” spanning tree of \( G \setminus v_f \). The \( MSTM \) is
composed of the following phases:

1. computes the base of the tree. This base of \( T' \) is the subtree of \( T \setminus v_f \) rooted at \( r \). It is
identified using a marking procedure (see below).
2. search extension. Using a “depth-first trip” [10] on \( T' \) each node \( v \) is visited until one of the
neighbor of \( v \) is not yet included in the current tree \( T' \).
3. connection is found. The visited node \( v \) finds a neighbor \( u \) that is not in the “on-building” tree.
Such a node is called a connection entry and \( v \) is named a connection node. Therefore, the edge \( (u,v) \) is added to \( T' \). The subtree of \( T \) containing \( u \) is reorganized in such a
way that \( u \) becomes its root using the root changing module (see below). Then, this subtree

\(^4\) Sons of \( v_f \) become “orphan nodes” or without “father”.

A Simple Distributed Algorithm for the Maintenance of a Spanning Tree
is included in $T'$ and its nodes are marked as nodes of $T'$. Now, the search of isolated subtrees is restarted on the extended $T'$.

Since $G$ is a 2-connected graph, when the “depth-first trip” is terminated all the nodes of $G \setminus v_f$ has been included in $T'$ and the algorithm terminated.

The full description of the maintenance module is given in the following. This module is composed of nine actions and uses two modules: The marking module and the root changing module. Such modules are shortly described below.

**Marking Module** ($MM(T; v_0; Stage, X, Y)$). After the execution of the algorithm, the label of each node $v$ has been changed into $Stage(v) = Y$. More precisely, assuming that each node has its variable “Stage” set to $X$ in $T$. The node $v_0$ starts the module. It modifies its stage: $Stage(v_0) = W$. At any time, a node $v$ with “Stage” equal to $W$ broadcasts a $m_{tok}$ message to all its sons in $T$. If $v$ receives such a message, its stage will be set to $W$ if it has sons. Otherwise, if it has no son or all its sons are marked: It marks itself $T$. all its sons in this son to its list of marked sons. The end of this module is detected by the root when it receives a $m_{back}$ message from each of its sons in $T$. The cost of this module when applied to a tree $T = (V, E)$ is $(#V - 1)\ m_{tok} + (#V - 1)\ m_{back}$ messages.

**Root Changing Module** ($RCM(T; r)$). This module will be used to change the root of a tree $T$. So if a node $r$ has to replace the root $v_0$, it starts the module: It sets its variable “Stage” to $W$ and sends a $rc_{tok}$ message to its father in the “old tree” \(^5\). At the reception of such a message by $v \neq v_0$, it changes its stage from $A$ to $W$ and sends a $rc_{tok}$ message to its father in the “old tree”. When $v_0$ is attained from $w$, its son in the “old tree”, $v_0$ sets its stage to $A$ and its father to $w$ and sends a $rc_{son}$ message to $w$. Then, the process is applied reversibly to all the nodes in the “to be modified path”. The $RCM$ uses, in the worst case, $(2#V - 2)$ messages when applied to a tree $T = (V, E)$.

Now, we present an implementation of the maintenance module in the asynchronous message passing model where $v_f$ crashes \(^6\).

---

**Maintenance Module**: $MSTM(G, T; v_f)$

- **Input**: A graph $G = (V, E)$ with a spanning tree $T = (V, E_T)$, and a maintainer node $v_f$.

- **Variables**:
  - Stage($v$) ∈ {A, Ended, F, SC, W}, Father($v$), Sons($v$), B($v$),
  - $T_{OldExplore}(v)$, Potential($v$), $T_{OldIgnore}(v)$, Crushed($v$).

- **Initialization**:
  - $\forall v \in V \setminus \{v_f\},\ Stage(v) = A$.
  - Sons(Father($v_f$)) = Sons(Father($v_f$)) \ {v_f}.
  - $\forall v \in Sons(v_f),\ Father(v) = w$.
  - $\forall v \in V,\ ToExplore(v) = Sons(v) \setminus Crushed(v)$.
  - $\forall v \in V,\ Potential(v) = \emptyset$.
  - $\forall v \in V,\ T_{OldIgnore}(v) = Crushed(v)$.

- **Result**: A spanning tree $T^\prime$ of $G \setminus v_f$ with root $r$. In this case, $\forall v \in V \setminus v_f$ is labeled Stage($v$) = Ended.

- **Actions**:

  1. **MST A1**: (For the initiator $r$ only; execute once)
     - Potential($r$) = B($r$) ∪ Sons($r$)
     - MM($T_f, r, Stage, A, F$); Stage($r$) = SC;

  2. **MST A2**: (For each node $v$ such that Stage($v$) = SC, execute once)
     - $\forall v \setminus v_f,\ Stage(v) = W$.
     - Let $p$ be a port number in Potential($v$),
     - Potential($v$) = Potential($v$) \ {p};
     - send($MSTM$, v, p);
     - $\forall v \setminus v_f,\ Stage(v) = W$.
     - Let $p$ be a port number in $T_{OldExplore}(v)$;
     - $T_{OldExplore}(v) := T_{OldExplore}(v) \ {p}$;

---

\(^5\)The subtree rooted at $v_0$ is included in $T'$. Since nine of maintenance module $X$ each node $v$ sends a $Y$ message to its father in the “old tree”. The end of this module is detected by the root when it receives the “to be modified path”. The $RCM$ uses, in the worst case, $(2#V - 2)$ messages when applied to a tree $T = (V, E)$.

Now, we present an implementation of the maintenance module in the asynchronous message passing model where $v_f$ crashes.

\(^6\)\(v_f\) is stored in the Crushed list of its neighbors.
It is the only one that executes action construction of the tree it belongs to.

Stage \((T\) after the initialization, \(v \in V_G\)) \(\forall v \in V_G\setminus v_f, Stage(v) = A\). The maintainer node \(r\) initiates the computation: It is the only one that executes marking action \(MSTA1\). So \(r\) executes the marking action. When \(r\) finishes the execution of this module, its subtree \(T_r\) in \(T\) is marked \(F\). It means that this subtree is included in the on-building tree. Node \(r\) switches to the “searching connection” phase \((Stage(r) = SC)\). Then, step by step, a tree \(T_r\) rooted at \(r\) is extended. At any time there is only one node in the “searching connection” phase and this node is in charge to extend the construction of the tree it belongs to.

Node \(v\) such that \(Stage(v) = SC\) with no empty “Potential” set starts the search of connection sending a \(mst_sc\) message via one of its potential ports (sub-action \(MSTA2(1)\)). If a node \(u\) is a connection entry \((Stage(u) = A)\) received this message, it executes action \(MSTA3(1)\) to prepare the extension of \(T_r\). So its subtree is added to \(T_r\) after a reorganization.

Let \(T(u)\) be a tree that contains the connection entry \(u\). Node \(u\) starts the execution of the root changing module. The marking module is then applied to the tree rooted at \(u\). Now, \(u\) is ready to include the tree \(T_r\), variable \(Father\) of \(u\) is set to \(v\) and a \(mst_son\) message is sent to \(v\). When a \(mst_son\) message has arrived at \(v\) from \(u\), node \(u\) is added in both \(Sons(v)\) and \(To_Explored(v)\) (action \(MSTA4\)). That is, \(T_r\) is extended and \(v\) continues the search of another connection entry.

At any time, the actions \(MSTA5\) and \(MSTA6\) are used to update the lists “Potential” and to prepare their computations using the lists “ToIgnore”. These actions may be executed concurrently with the other actions. If \(Stage(v) = F\) then \(u\) cannot be a candidate for reconnection since it is yet in the tree and so it must be avoided from the lists “Potential” of its neighbors. Moreover, the neighbors of \(v\) yet in the tree (their variables \(Stage\) are valued \(F, W, SC\)) has to be ignored in its “Potential”.

When a node \(v\) is in “searching connection” phase and has an empty list “Potential”, which arises when it has no neighbor as connection entry \((A)\) and all it neighbors are marked \(F\) have avoided themselves from its list “Potential", \(v\) executes action \(MSTA2(2)\) to mean that it does not succeed to find an extension for \(T_r\). So one of its sons, if possible, is chosen to become in the “searching connection” phase. To do this, \(v\) sends a \(mst_deleg\) message to this son. The list of the candidate
nodes to a reconnection from the “chosen” son is computed (“Potential”) executing action \textit{MST A} followed by action \textit{MST A}.

Eventually, some node $v$ in $T'$ has extended its subtree as much as possible (\textit{Stage}($v$) = \textit{Ended}), so it informs, with action \textit{MST A}(3) (or \textit{MST A}(9)), its father sending a \textit{mstended} message. At the reception of this message, the father can proceed the extension search from one of its other sons executing action \textit{MST A}.

Furthermore, only $r$ detects the end of the extension search (\textit{Father}($r$) = \textit{\bot}): When \textit{Stage}($r$) = \textit{Ended} a largest subtree rooted at $r$ of $G \setminus v_f$ has been computed.

4. PROOF OF CORRECTNESS AND ANALYSIS OF THE \textit{MOST} ALGORITHM

We recall that after the detection of a crash of a node $v_f$ in $G$ by the \textit{crash detection module} (\textit{CDM}), the initial spanning tree of $G$ is disconnected. The goal of our algorithm is to rebuild a spanning tree of $G \setminus v_f$. Now, we show the correctness of the \textit{MOST} algorithm using a scheme based on the modules properties and the graph connectivity. We use \textit{MOST} instead of \textit{maintenance of a spanning tree}. Our proofs assume that when the modules are executed in some action they are executed as atomic steps with the two following consequences:

- Each module does its expected task,
- Each module terminates during the execution of the action referencing it.

We will base our proof and analysis on the main module of our algorithm: The \textit{maintenance module}. The analysis will be closed with space and time complexity measures.

\textbf{Lemma 4.1} Let $G$ be a 2-connected graph and $T$ be a spanning tree of $G$ rooted at $v_0$. Then the tree $T_r$ rooted at $r$ (the maintainer node) built by the \textit{MSTM} is a spanning tree of $G \setminus v_f$.

\textbf{Overview of the Proof.} The proof is by induction on the number of disconnected subtrees. We consider the two following cases. After the crash of a node $v_f$, the tree $T$ is disconnected. We denote by $\#\text{strees}$ the number of disconnected subtrees of $T$ such that:

- \textit{case1}: crashed node is the root $v_0$, $\#\text{strees} = \#\text{Sons}(v_0)$,
- \textit{case2}: crashed node is not the root $v_0$, $\#\text{strees} = \#\text{Sons}(v_f) + 1$.

We must show that after such event, the \textit{maintenance module} will reconnect such subtrees to construct a spanning tree $T'$ of $G \setminus v_f$. We define the property $C$ on a graph $G$ and a given node $v$ as: $(G, v)$ satisfies $C$ iff $G \setminus v$ is a connected graph. Since $G$ is a 2-connected graph and then $v_f$ is not a “cut-node”, the property $C$ is verified for $(G, v_f)$. If $\#\text{strees} = 1$: For case1, the root has only one son and the succorer son of $v_0$ becomes without any change the root of a spanning tree of $G \setminus v_f$. The algorithm works correctly since starting from a graph $G$, the succorer son of $v_0$ executes action \textit{MSTA}1 to mark its subtree, followed by action \textit{MSTA}2(3) to detect the end of the maintenance procedure. In case2, $v_f$ is a leaf and then $T \setminus v_f$ is a spanning tree of $G \setminus v_f$. The tree rooted at $v_0$ remains a spanning tree of $G \setminus v_f$. As previously, the algorithm works correctly.

The induction property is stated as follows. \textit{Let $G=(V,E)$ be a graph and $v_f \in V$ a crashed node. Then, for all spanning tree $T$ of $G$, if $T \setminus v_f$ is disconnected into less than $l \geq 1$ subtrees then the \textit{MSTM} builds a spanning tree of $G \setminus v_f$.}

Now, suppose that the algorithm works correctly for $l \geq 1$. So let $T$ be a spanning tree of $G$ such that $T \setminus v_f$ is splitted into $T_0, \ldots, T_l$ disconnected subtrees rooted respectively at $w_0, \ldots, w_l$. Let $T_i = (V_i, E_i)$. For case1, $w_0$ is assumed to be the succorer son of $v_f$. For case2, we can assume that $w_0 = v_0$. So the \textit{MSTM} algorithm starts with $w_0$ as a maintainer node (\textit{MSTA}1). Since $G$ is a 2-connected graph, there are $v_0 \in V_0$ and $v_f \in V_f$ such that $w_0$ is a connection node and $w_f$ is its corresponding connection entry. These nodes are detected during the “depth-first trip” on $T_0$ induced by the application of the actions \textit{MSTA}2(2), \textit{MSTA}4, \textit{MSTA}7, \textit{MSTA}8, \textit{MSTA}9.
Then, the actions $MSTA3(1)$ and $MSTA4$ are applied. These actions modifies $T_0$ adding a subtree $T_j = (V_j, E_j)$ which is the result of the application of the $RCM$ to $T_j = (V_j, E_j)$ with a new root $w_j$.

Let $T_0'$ be the subtree $T_0$ augmented by $T_j$ and the edge $(w_0, w_j)$. Let $T' = (V, E')$ be the spanning tree of $G$ where $E' = (E_T \setminus (E_j \cup (v_f, w_j))) \cup (E_j \cup (w_0, w_j))$. Thus, consider the application of the $MSTM$ on $G$ and $T'$ when a crash of $v_f$ occurred. Then the tree $T'$ is splitted into exactly $l$ disconnected subtrees. Hence by the induction hypothesis the $MSTM$ succeeds to maintain a spanning tree of $G \setminus v_f$.

The algorithm uses the same actions than the algorithm applied to $G$ and $T$. It is easy to see that the “depth-first-trip” leads the algorithm to treat the node $w_0$ since it has not found a connection before. Otherwise, $w_0$ would be a connection node when the algorithm is applied to $G$ and $T$.

So the application of the $MSTM$ to $G$ and $T'$ leads to the same structure than $MSTM$ applied to $G$ and $T'$ after the fusion of $T_j$ in $T_0$. The application of the $MSTM$ to $G$ and $T'$ leads to the same result than $MSTM$ applied to $G$ and $T'$. Since the execution of the $MSTM$ on $G$ and $T'$ succeeds the execution of the $MSTM$ on $G$ and $T'$ succeeds.

For the message complexity, we will show in the following the worst case of the execution of the $MSTM$:

1. The action $MSTA1$ can only be applied once. So its cost is due to the application of the marking module to the nodes of the subtree rooted at $r$. At most $(2\#V_T - 2)$ messages.
2. The action $MSTA2$ is applied by each node of the graph $G \setminus v_f$ such that its “Stage” variable is set to $SC$. The application of this action involves, in the worst case, $c_1$ $mst_{sc}$ messages, $c_2$ $mst_{deleg}$ messages, such that $c_1 + c_2 = \deg(G) - 1$, and one $mst_{ended}$ message. So, in the worst case, the total cost of the action $MSTA2$ is $2\#E$ messages.
3. The action $MSTA3$ can be also applied only once for each node of the subtrees rooted at the sons of $v_f$ different from $r$. So its cost is due to the application of the marking module to each of these subtrees augmented by the cost of the root changing module. The number of calls of this modules is exactly the number of sons of $v_f$. The modules cost is at most $(\sum_i \#Sons(v_f))(4\#V_T - 4)$ where $v_i \in Sons(v_f)$: Adding the cost of $MSTA1$, this part of the action is bounded by $4\#V - 4\#Sons(v_f)$. At each reconnection one $mst_{son}$ message will be sent, otherwise one $mst_{fail}$ will be sent if the reconnection failed. At most the rule $MSTA3$ uses $(4\#V - 4\#Sons(v_f)) + (#Sons(v_f) - 1) + (#E - #V - 2 - #Sons(v_f))$ messages.
4. The action $MSTA4$ does not involve messages.
5. The action $MSTA5$ is applied once by each node to update the “To Ignore” and the “Potential” sets of its neighbors not in the tree. So, a priori, the cost of the application of this action is $2\#E$, but since we use the list “To Ignore” only $\#E$ applications are necessary. Moreover, the edges of the tree have not to be taken into account and also the sons of $v_f$. Finally, the cost of the use of $MSTA5$ is $\#E - \#V - 2 - #Sons(v_f)$ messages.
6. Both the actions $MSTA7$, $MSTA8$ do not involve messages.
7. The action $MSTA9$ sends one $mst_{deleg}$ message and one $mst_{ended}$ message over the edges of the tree without $v_f$ (induced by the use of the “depth-first-trip”). Thus, this action uses at most $2(\#V - 2 - #Sons(v_f))$ messages.

Then,

Lemma 4.2 Let $G = (V, E)$ be a graph. If the $MSTM$ is applied to $G$ then it uses at most $4(\#E + \#V)$ messages.

For the space requirements, each node $v$ has the following local variables. They may be decomposed into two components:

1. $B(v)$, $Sons(v)$, $Feedbacks(v)$, $ToExplore(v)$, $Potential(v)$, $ToIgnore(v)$, $Crashed(v)$,
2. $Stage(v)$, $Father(v)$, $Traversed$, $Suc(v)$.
Thus, to encode the first component every node needs to maintain subsets of its neighbors as descendants, for the set of sons for example. So, every node $v$ needs $7 \deg(v)$ bits to store this component, where $\deg(v)$ is the degree of $v$. By taking into account the other variables used in the second component, we can claim that the total memory requirement of each node $v$ is $7 \deg(v) + \log_2 7 + 3 \log_2 2$. bits. Then,

**Lemma 4.3** The **MOST** algorithm uses at most $O(\deg(G))$ bits per node when applied to a graph $G$.

The cost of the **MOST** algorithm is those of the maintenance module adding the cost of the failure detection module and the simple path propagator module. The last module is not invoked when the crashed node is the root.

To analyze the time complexity of algorithms based on graph exploration on asynchronous networks, we use a level number and a diameter defined in Section 2.1 to help us in the proofs of the worst case bound. We associate with leaves of a tree $T$ level $l_0 = 0$ and the level of each node is the maximum level number of its sons plus 1. This number will be computed during the construction of the spanning tree of $G$ using the spanning tree module (see Section 2.3). So the message $st\_back$ is replaced by the level number. We introduce the function $To\_number(msg)$ (resp. $To\_message(level)$) to convert the $st\_back$ message (resp. the level number) to a level number (resp. to an $st\_back$ message). Thus, rules $STA2$ and $STA4$ of the spanning tree module are modified such as:

- The level number $level(v)$ is initialized to 0 in $STA2(1)$. Then, the message $To\_message(level(v))$ is used as a substitute for $st\_back$ message.
- $STA4$. (A message $<st\_back>$ has arrived at $v$ from port $q$)
  - $ Terminated(v) \leftarrow Terminated(v) \cup \{q\}$
  - $ level(v) \leftarrow max(level(v), To\_number(st\_back))$;
  - if $(included(v) \neq B(v)$ and $ Terminated(v) \subseteq Sons(v))$
    - $ Stagemy(v) \leftarrow Y$
  - $ level(v) \leftarrow level(v) + 1$;
  - $ x \leftarrow included(v) \cap Sons(v)$
  - send<$To\_message(level(x))$> via port $Father(v)$;

The two following lemma follows the definition and the computation of the level:

**Lemma 4.4** Given a graph $G$ with diameter $D(G)$, the maximum node level number assigned during the execution of the $STM$ is bounded by $D(G)$.

**Proof.** Initially, let $T_0$ be the tree containing the leaves of the tree to be build. Every leaf node $v_l$ has level $level(v_l) = 0$. At this step, the leaves are discovered (rule $STA2$). The tree $T_1$ is obtained after the local termination of the nodes of level $l - 1$. This step involves the send of their level numbers (rule $STA4(2)$). So, we can do this at most $D(G)$ times before we get a spanning tree of $G$. Thus, the maximum level number obtained is $D(G)$.

**Lemma 4.5** During the execution of the $STM$ on the graph $G$, the node at level $l$ will have sent a $st\_back$ message within $D(G) + l$ time.

**Sketch of the Proof.** Let $t_l$ be the time at which all the nodes of $G$ are included in the on-building tree $T$ (labeled $WA$). Then, by induction we show that each node $v$ at level $l$ in the graph, $v$ will sent an $st\_back$ message within $t_0 + l$ time.

From the two previous lemmas, we deduce:

**Lemma 4.6** The execution of the $STM$ on the graph $G$ uses at most $2D(G)$ time.

Using the same proofs, we claim the following:

**Lemma 4.7** Given a tree $T = (V, E)$. The application of the marking module ($MM$) to $T$ uses at most $2D(T)$ time.
Lemma 4.8 The execution of the root changing module (RCM) on a tree \( T = (V, E) \) requires at most \( 2D(T) \) time.

Now, we study the time complexity of the main module: \( MSTM \). We use the same reasoning as such used for the message complexity.

Lemma 4.9 Given a spanning tree \( T = (V, E_T) \) of a 2-connected graph \( G = (V, E) \) rooted at \( v_0 \). The maintenance module takes at most \( 6\#V - 2\#V_T \) time.

Proof. First, \( v_0 \) applies the \( MM \) to construct the base of the “on-building” tree. Second, we know that each subtree resulting after the disconnection of \( v_f \) is traversed sequentially because of the use of the “depth first” exploration technique. So without loss of generality, each subtree applies one time the \( RCM \) followed by the \( MM \). Let \( u \) be a son of \( v_f \). The number of traversed edges is bounded by \( 2(\#V - 2) \) for the reconnection phase. Thus, the total time used by the \( MSTM \) is a priori bounded by \( 4(\sum_{i=2}^{\#Sons(v_f)} D(T_{v_i})) + 2D(T_r) + 2(\#V - 2) \), such that \( v_i \in Sons(v_f) \). The sum of the diameters of all the subtrees is bounded by \(#V\), then, the maximal tree construction module requires time bounded by \( 6\#V - 2\#V_T - 4 \).

Now we state our main result. The \( MOST \) algorithm takes at most \( GST + \#V + 6\#V - 2\#V_T \) time and uses at most \((GST + \#V + 4(\#E + \#V)) \) messages and \( 7 \deg(v) + \log_2 7 + 3 \log_2 2 \) bits per node. Then without taking into account the extra messages and the delay caused by the crash detection module, the following result completes the analysis.

Theorem 4.10 Let \( G = (V, E) \) be a 2-connected graph and \( T \) be a spanning tree of \( G \). After the crash of some node \( v_f \), the \( MOST \) algorithm maintains a spanning tree of \( G \setminus v_f \) in \( O(\#V) \) time using \( O(\#E + \#V) \) messages and \( O(\deg(G)) \) bits per node.

5. MAINTENANCE OF A SPANNING TREE OF \( K \)-CONNECTED GRAPHS AND \((K - 1)\) FAILURES

Now we propose an extension of our protocol to deal with \( k \)-connected graphs. Thus, we propose a protocol to tolerate at most \( k - 1 \) consecutive failures. That is, if a graph is \( k \)-connected, the previous algorithm would be applied iteratively to treat each failure occurrence. As presented above, the algorithm works correctly without any more assumption except the information about the succoror son after each failure occurrence: The succoror son of the actual tree resulting after this crash. So, adding an update of the succoror son of such a tree, the protocol may be extended to deal with the maintenance of a spanning tree of \( k \)-connected graph tolerating \( k - 1 \) consecutive failures.

To treat each failure occurrence, our distributed algorithm, referred to as \( MOST_k \) algorithm, consists on the following phases:

1. detection of the crash. Applying the crash detection module.
2. propagation. Applying the simple path propagator module if the crashed node is not the root.
3. maintenance update. Maintainer node starts the execution of the maintenance update phase.
4. cleaning. To deal with the next failure occurrence using the previous three phases, the used local variables require some initialization. For example, all the nodes will set their variables \( Stage \) to \( A \) an so on. It is possible using an extended version of the marking module without adding extra complexity.

To achieve the maintenance update phase, we use a new module denoted by “\( MSTM \)” for maintenance update module. That is, for a graph \( G \), at each step \(^7\), the \( MSTM \) does the following:

- It builds a spanning tree of \( G \) deprived of the crashed node using the \( MSTM \).

\(^7\)The step denotes a failure occurrence.
Then the new root checks if it has a succorer son:
- if the root is newly defined, it deactivates its status as the succorer son of the crashed root, and it chooses one of its sons to become its succorer son.
- if the crashed node is a succorer son, then the root chooses, if it's possible, another son to become its succorer son.
- otherwise, the maintenance module doesn't need to update the succorer son.

Assuming that the time between each consecutive failure occurrence is greater than the time needed to treat each failure,

**Theorem 5.1** Let $G$ be a $k$-connected graph where $k \geq 2$. The $\text{MOST}_k$ algorithm constructs a spanning tree of any graph $G \setminus H$ such that $H$ is a subgraph of $G$ and $\#V_H \leq (k - 1)$.

**Proof.** We will prove this theorem by induction on the number of failures. Let $P$ be a property that at any time, the root has its succorer son. That is, after the first crash, let $v_f$ be the crashed node, the protocol works as mentioned above to update the spanning tree of $G \setminus v_f$, adding the maintain of the property $P$. This property requires an updating if $v_f$ is the root which consists on associating a succorer son with the succorer son of $v_f$. We denote by $s_i$ the step corresponding to the failure $i$ and by $r_i$ the root of the corresponding spanning tree denoted by $T_{r_i}$. At step $s_0$, $T_{r_0}$ is the root of $G \setminus \emptyset$. Only $r_0$ admits a succorer son. We assume that a pre-chosen son of $r_0$ denoted $r_1$ is identified to become the succorer son of $r_0$.

At the next step failure $s_1$, if the crashed node is $r_0$ then the protocol updates a spanning tree of $G \setminus r_0$. So, the tree $T_{r_1}$ is a spanning tree of $G \setminus r_0$ and some son of $r_1$ is chosen to become its succorer son ($r_2$). Otherwise, if the crashed node is not $r_0$, we have two cases. If $r_1$ crashes, then $r_0$ chooses another son to become its succorer son. Otherwise, the property $P$ remains verified without any requirement.

Now, we suppose that at step $s_l$ such that $l < k - 1$, the protocol works correctly. To prove that the protocol works correctly for the next failure occurrence, we will consider the application of the algorithm on the resulting graph. Let $H$ be a subgraph of $G$ such that $\#V_H = k - 1$. The protocol works correctly for any subgraph of size less that $k - 1$. Let $Q$ be a subgraph of $H$ such that $\#V_Q = k - 2$. We denote by $T(k - 2)$ the spanning tree of $G \setminus Q$. To show that the $\text{MOST}_k$ algorithm is able to build a spanning tree of $G \setminus H$, it suffices to consider the application of the algorithm on $G \setminus Q$ after one failure. Therefore, the algorithm works correctly and the theorem holds.

**Corollary 5.2** Let $G = (V, E)$ be a $k$-connected graph and $T$ be a spanning tree of $G$. After the crash of some set of nodes $V_H$ such that $\#V_H \leq k - 1$, the $\text{MOST}_k$ algorithm maintains a spanning tree of $G \setminus H$ in $O(\#V_H \times \#V)$ time using $O(\#V_H \times (\#E + \#V))$ messages and $O(\text{deg}(G))$ bits per node.

**Remark.** Our algorithm works correctly for complete graphs tolerating any number of failures, since the graph remains connected.

### 6. CONCLUSION AND FUTURE WORKS

The main focus of this work has been to design a distributed protocol which executes on $k$-connected networks to reorganize a spanning tree within those networks. The maintenance of this structure (spanning tree) by incremental updates is studied in unreliable networks in the presence of crash failures.

The paper has two major contributions. First, we present a new formalization of such a problem using an occurrence of the Menger's theorem and its relation with the possible construction of a spanning tree. The notion of succorer son is also introduced. Second, the protocol is

---

8 All the nodes of the graph $G$ are alive nodes.
A Simple Distributed Algorithm for the Maintenance of a Spanning Tree

implemented and analyzed in the asynchronous message passing model. To present easily the last contribution, we divided our protocol on a set of phases achieved using a set of modules. The algorithm tolerates \( k - 1 \) consecutive failures when applied to a \( k \)-connected graph. It’s based on an incremental application of the maintenance-update module. So at any step, this module is responsible for both maintaining of the spanning tree and for updating the succorer son. For a given graph \( G = (V, E) \), after each failure occurrence, our algorithm maintains a spanning tree of the resulted graph and computes the succorer son of its root in \( O(\#V) \) time using at most \( O(\#E + \#V) \) messages and \( O(\text{deg}(G)) \) bits per node.

The connectivity of a network can be used to measure the network’s reliability. Our protocol is based on the knowledge of spacial connectivity structures within the network which leads to optimization of distributed control applications. But, for a \( k \)-connected network if more than \( k - 1 \) crashes occur, the network is split and partitioned into many components. In this case only one connected component is taken into account by our algorithm. It would be desirable to maintain a dynamic forest of trees in such networks.

Our approach assumes that each process is equipped with an unreliable local failure detector, with a minimum required properties to solve a consensus problem [4]. Even, such failure detectors guarantee the reach of “stable” periods: During such periods, the failure detectors are accurate. So, our applications work during these periods and such assumption may only make our applications delayed. Another extension is to deal with the case of unreliable failure detectors without assumptions. So, protocol to maintain spanning tree starts before the reach of the stabilization periods. We are interested to design approximation algorithms to deal with the maintenance problem with errors.

REFERENCES