Fault-Tolerant Reconfiguration of Trees and Rings in Networks *

Anish Arora
Computer Science
Ohio State University
Columbus, OH 43210

Ashish Singhai
Computer Science
University of Illinois
Urbana-Champaign, IL 61801

Abstract

We design two protocols that maintain the nodes of any computer network in a rooted spanning tree and in a unidirectional ring, respectively, in the presence of any finite number of fail-stop failures and repairs of network nodes and communication channels. Our protocols are fully distributed, have optimal time and space complexity, and illustrate two different methods for the design of nonmasking fault-tolerant protocols.

Keywords: Network operating systems, reliability, fault-tolerance, methodologies, distributed programming, graph algorithms.

1 Introduction

Cooperation between the nodes of computer networks is commonly realized by organizing the nodes into a convenient logical structure such as a ring, a star, or a tree. Such cooperation, however, poses a problem if faults can occur during the computation of the computer network: Fault occurrences can perturb the logical structure and thereby necessitate a reorganization of the nodes.

An ideal solution to the problem of reorganizing network nodes into the desired logical structure is to design a protocol that is "nonmasking fault-tolerant". Nonmasking fault-tolerant protocols make no assumptions about the predictability of fault occurrence and, hence, tolerate further perturbations to the structure that may occur while they reorganize the nodes. We call these protocols nonmasking as they make no attempt to mask the perturbed structure from the cooperative mechanisms that rely on the structure.

Remarkably, methods for the design of nonmasking fault-tolerant protocols have received little attention. This is largely due to the supposition that such methods will be complex, and will explicitly consider all combinations of the number, time, and duration of faults occurrences. Our experience shows, however, that this supposition is false. We find [1-3] that a few, simple methods suffice for the effective design of most nonmasking fault-tolerant protocols, and avoid the combinatorial problem noted above. Below, we discuss two such methods for design of protocols to reorganize network nodes.

One method is to compute the set X of all structures that result from perturbing the desired structure x by 0, 1, 2, ... simultaneous faults. Then, derive a protocol so that starting from any protocol state where the structure is in X, subsequent computation of the protocol (i) always yields protocol states where the structure is in X and (ii) reaches within a bounded number of steps a protocol state where the structure is x. The correctness of this method follows from the fact that even if faults occur while the protocol is executing then, by (i), the protocol is always in a state where the structure is in X; hence, subsequent computation of the protocol is guaranteed, by (ii), to reach a state where the structure is x.

Another method is to make no assumption about the structure that results from the execution of faults.

[0]Supported in part by NSF grant CCR-9308640 and OSU Grant 221506

0-8186-6685-4/94 $4.00 © 1994 IEEE

Instead, ensure that the protocol is "stabilizing", in the following sense. Starting from an arbitrary protocol state, subsequent computation of the protocol converges within a bounded number of steps to a protocol state where the structure is x. Correct execution of the computer network then resumes, and continues until a subsequent fault occurrence perturbs the structure, in which case the cycle of convergence repeats.

Overview of the paper. We formulate the two methods described above in terms of a uniform definition of fault-tolerance [1-2]. Using these methods, we design two fully distributed, nonmasking fault-tolerant protocols that maintain the nodes in an arbitrary but connected computer network in a rooted spanning tree and in a unidirectional ring, respectively. More specifically, our protocols tolerate any finite number of fail-stop failures and repairs of nodes and channels in the computer network.

Our protocols are optimal in their time and space complexity: Both reach a fixpoint within $O(N)$ time, where N is the number of network nodes that are "up" (i.e., currently not stopped due to a fail-stop failure). Our protocol for ring maintenance, infact, uses as a basis any nonmasking fault-tolerant protocol for maintaining a rooted spanning tree, and reaches a fixpoint within *constant* time of the latter reaching a fixpoint. Both protocols use $O(log\ M)$ space at each node and in each message, where M is the total number of nodes.

Our protocol for ring maintenance is optimal in another sense: The ring it establishes spans at most $2(N-1)$ communication channels and the number of channels between successive nodes in the ring is at most 3. Both these "length" and "dilation" bounds are optimal since there exist networks that have no ring embedding of length less than $2(N-1)$ or dilation less than 3. (For example, consider respectively nodes organized in a line and in a 3 by 3 mesh with any two parallel lines deleted that are a unit distance apart.)

The rest of this paper is organized as follows. In Section 2, we state our assumptions and give a formal definition of protocols and their nonmasking fault-tolerance. In Section 3, we use the first method discussed above to present a nonmasking fault-tolerant protocol for maintaining a rooted spanning tree. We build upon this protocol, in Section 4, to present a stabilizing protocol for maintaining a unidirectional ring. In Section 5, we discuss related work and make some concluding remarks.

2 Preliminaries

Assumptions. A computer network consists of M nodes, that are named $1, ..., M$, and channels, that each connect a unique pair of nodes. At any instant, each node and channel is either "up" or "down". Only up nodes can execute actions to maintain the desired structure. Actions of an up node may involve communication only with up nodes that are "adjacent" to that node; i.e., connected by up channels. Channels are bidirectional. We henceforth use interchangeably the terms "node" and "up node" and the terms "channel" and "up channel".

Fail-stop faults can arbitrarily change the set of nodes and channels, as long as the nodes remain connected. (We assume connectivity of nodes for simplicity alone: If nodes become partitioned, then the nodes of each partition will be organized into the desired structure.) Repairs can likewise arbitrarily change the set of nodes and channels. Repairing a node may involve initializing its state appropriately, and repairing a channel may involve initializing the state of its incident nodes.

Protocols. A "protocol" is a set of "variables" and a finite set of "actions". Each variable has a predefined nonempty domain. Each action has the form:

$$\langle \text{guard} \rangle \ \longrightarrow \ \langle \text{statement} \rangle$$

A guard is a boolean expression over the protocol variables. A statement updates zero or more protocol variables and always terminates upon execution.

Let p be a protocol. A "state" of p is defined by a value for each variable in each node of p (chosen from the domain of the variable). An action is "enabled" of p at a state iff the guard of the action holds at that state.

A "state predicate" of p is a boolean expression over the variables of p. A state predicate S is a "fixpoint" of p if for each state of p where S holds, no action of p is enabled in that state. A state predicate T is "preserved" by an action of p iff executing the action, starting from any state of p where the action is enabled and T holds, yields a state where T holds. A state predicate T is "closed" in p iff T is preserved by each action of p.

A "computation" of p is a fair, maximal sequence of steps; in every step, some action in p that is enabled in the current state is executed. Fairness of the sequence means that each action in p that is contin-

uously enabled along the sequence is eventually executed. Maximality of the sequence means that if the sequence is finite then no action in p is enabled in the final state.

Finally, a state predicate Q "converges to" R in p iff Q and R are closed in p and, starting from any state where Q holds, every computation of p has a state where R holds.

Nonmasking Fault-Tolerance. Let p be a protocol that maintains a structure x in a computer network. We first observe that the set of "fault-free" states of p , i.e. the states where the structure is x, can be characterized by a state predicate S that is a fixpoint of p. Such a state predicate identifies the "invariant" of p.

We next observe that the set of states that p reaches in the presence of faults can be characterized by a state predicate T that is closed in p. Such a state predicate identifies the "fault-span" of p. Note that the fault-span includes the fault-free states of the protocol. (Hence $S \Rightarrow T$.) Examples showing how to design fault-span predicates appear in [2]. These examples take the view that all classes of faults can be represented as actions that change the protocol state. For instance, a fail-stop failure of a node can be represented as an action that marks an up node as being down, and a repair can be represented as an action marks a down node as being up and reinitializes the state of that node.

We are now ready to give a uniform definition of fault-tolerance. Let S be the invariant and T be the fault-span of p (hence, S is a fixpoint of p and T is closed in p). And let F be a set of fault actions. For p to be $F-$tolerant, the following two requirements should be satisfied :

- Closure: T is preserved by the actions of F ,
- Convergence: T converges to S in p .

The definition above enables a formal classification of masking and nonmasking fault-tolerance [1-2]. Let p be F-tolerant. If $T = S$, we say that p is masking $F-$tolerant. Else ($T \neq S$), we say that p is nonmasking $F-$tolerant.

We observe finally that the first method discussed in Section 1 is formulated in terms of this definition by requiring that T implies that the structure is in the computed set X; and the second method T is formulated by requiring that T is the state predicate *true*.

3 Maintaining a Rooted Spanning Tree

In this section, we employ the first of the two methods discussed in Section 1 to derive a nonmasking fault-tolerant protocol for maintaining a rooted spanning tree. Recall that in this method we first compute the set of all structures that result from perturbing the desired structure, in this case a rooted spanning tree, by any finite number of simultaneous occurrences of fail-stop failures and repairs.

Let us represent the rooted spanning tree in terms of a "parent" relation between nodes; each node j maintains the index of its current parent node in the tree. Now, if any number of nodes or channels fail-stop, the resulting graph of the parent relation of the up nodes is a forest of trees. Moreover, if any number of nodes or channels repair, the resulting graph of the parent relation of the up nodes is also a forest, provided each node initializes itself to be its own parent when it repairs, and leaves its parent unchanged when a channel incident at it repairs.

We therefore propose to let the collection of all forests be the set X of structures that is preserved by the protocol we derive.

To derive our protocol, we need to address two problems. First, how to handle trees that are not rooted, that is, trees with nodes whose parent node or "parent channel" (i.e., channel to the parent node) are down. And second, how to handle multiple rooted trees in the graph of the parent relation of up nodes.

To solve the first problem, we need a mechanism to correct each node that has a down ancestor node or channel. We propose to correct such nodes, while maintaining the structure to be in X, by associating a color with each node, as follows. As long as all ancestor nodes and channels of a node j are up, the color of j is green. When a green colored j detects that its parent node or channel is down or is colored red, j colors itself red. Also, when a red colored j detects that it has no remaining children, it disowns its parent and elects itself root; to do so, it makes itself its own parent and colors itself green. The net effect of these coloring actions is that when any node or channel goes down (i) all descendents of that node or channel color themselves red and then (ii) all descendents of that node respectively disown their parent.

Observe that a node or channel, whose fail-stop makes its children color themselves red, may repair

before the children recolor themselves green. In this case, we require that upon repair the node or channel in question colors itself red.

To solve the second problem, we need a mechanism to merge trees. We propose to merge trees by giving precedence to the tree whose root has the highest index, as follows. Each node j maintains a root value denoting the index of the node that j considers to be the root of the spanning tree containing j. If the root value of j is lower than the root value of an adjacent node k, j merges into the tree containing k: j adopts the root value of k as the root value of j and makes k the parent of j. The net effect of these merge actions is that (i) the root values of the nodes along all tree paths from their root to their leaves is always nonincreasing and (ii) eventually the root value of each node is the index of the root of its tree.

Observe that merge actions should be allowed to execute between nodes j and k only if the color of both nodes is green and the color the edge (j,k) is green. For, if j merges with k when k is colored red or (j,k) is colored red, j will subsequently have to disown k. And, if j merges with k when j is colored red, the resulting state may have a red colored node whose ancestors are all up.

Our derivation above suffices to ensures that, starting from any structure in X, the node with the highest index will elect itself as a root and color itself green via a coloring action, following which the remaining nodes will join its tree via merge actions.

An informal description of our protocol now follows. Node j maintains the parent, root, and color values in variables $par.j$, $root.j$, and $d.i$, respectively. Let $Adj.j$ be the set of nodes adjacent to j. The node has three actions: The first action is used to color j red; as described above, this action is enabled when $par.j \notin Adj.j \cup \{j\}$, $col.(par.j) = red$, or $col.(j,(par.j)) = red$. The second action is used to disown the parent of j; as described above, this action is enabled when each node $k \in Adj.j$ satisfies $par.k \neq j$. The last action is used to merge j into the tree containing k; as described above, this action is enabled when $k \in Adj.j$, $root.j < root.k$, and $col.j = green \wedge col.k = green \wedge col.(j,k) = green$.

Remark on notation : We use "parameters" to specify the third action. A parameter ranges over a finite domain. Its function is to enable a set of actions to be represented as one parameterized action. For example, let m be a parameter whose value is 0, 1 or 2; then the parameterized action $act.m$ abbreviates the following set of three actions.

$act.(m := 0)$ ▯ $act.(m := 1)$ ▯ $act.(m := 2)$
(End of remark.)

Our protocol, RST, for maintaining a rooted spanning tree therefore consists of the following three actions for each node:

```
node       j (j : 1..M)
var        root.j, par.j, : 1..M;
           col.j : {green, red};
parameter  k : 1..M;

begin
   col.j = green ∧
   (col.(par.j) = red ∨ col.(j, (par.j)) = red
                      ∨ par.j ∉ Adj.j ∪ {j})
⟶
   col.j := red
▯
   col.j = red ∧
   (∀k : k ∉ Adj.j ∨ par.k ≠ j)
⟶
   col.j, col.(j, par.j), root.j, par.j := green, green, j, j
▯
   col.j = green ∧
   k ∈ Adj.j ∧ col.(j, k) = green ∧
   col.k = green ∧ root.j < root.k
⟶
   root.j, par.j := root.k, k
end
```

Figure 1 : Protocol RST

We illustrate Protocol RST by an example. Figure 2(a) shows a 5-node network at an invariant state where node 4 is the root of the spanning tree, since node 5 is down. Consider, now, that node 4 and channel (1,2) fail. In this scenario, nodes 1 and 2 color themselves red by their first actions, since their parent channel and parent node, respectively, are down. Subsequently, node 3 colors itself red by its first action when it detects that its parent, node 2, is colored red (Figure 2(b)). Since nodes 1 and 3 have no children they can elect themselves as root by their second actions (Figure 2(c)), and since node 3 is the highest up node, nodes 1 and 2 will join its tree by their third actions. If node 5 now repairs, it will color itself red (Figure 2(d)), and since it has no children it will eventually color itself green by its second action, after

which all nodes will join its tree by executing their third actions (Figure 2(e)).

In the rest of this section, we show that RST is nonmasking fault-tolerant of fail-stop failures and repairs. We begin by characterizing the fault-span predicate T and the invariant predicate S. Let $T =$

$$
\begin{array}{l}
\text{graph of parent relation of up nodes is a forest} \\
\wedge\ (\forall j:\ j \text{ is up } \Rightarrow \\
\quad (par.j = j \ \Rightarrow\ root.j = j) \ \wedge \\
\quad (par.j \neq j \ \Rightarrow\ root.j > j) \ \wedge \\
\quad (col.j = red \ \Rightarrow\ (col.(par.j) = red \\
\qquad \vee\ col.(j,(par.j)) = red \\
\qquad \vee\ par.j \notin Adj.j \cup \{j\})) \ \wedge \\
\quad (par.j \in Adj.j \ \Rightarrow\ (root.j \leq root.(par.j) \\
\qquad \vee\ col.(par.j) = red \\
\qquad \vee\ col.(j,(par.j)) = red)) \ \wedge \\
\wedge\ (\forall j,k:\ (j,k) \text{ is up } \Rightarrow \\
\quad (col.(j,k) = red \ \Rightarrow \\
\qquad (p.j = k \wedge (col.j = red \vee root.j > root.k)) \\
\qquad \vee\ (p.k = j \wedge (col.k = red \vee root.k > root.j)))))
\end{array}
$$

Let $S =\ T\ \wedge$

$$
\begin{array}{l}
(\forall j:\ j \text{ is up } \Rightarrow \\
\quad (col.j = red \ \Leftarrow\ (par.j \notin Adj.j \cup \{j\} \\
\qquad \vee\ col.(par.j) = red \\
\qquad \vee\ col.(j,(par.j)) = red) \ \wedge \\
\quad col.j = green) \ \wedge \\
\quad (\forall k : k \notin Adj.j \ \vee\ root.j = root.k))
\end{array}
$$

Lemma 1 *At each state of RST where S holds, the graph of the parent relation is a rooted spanning tree.*

Lemma 2 *S is a fixpoint of RST.*

Lemma 3 *T is closed in RST.*

Theorem 1 *(Closure) T is closed under fail-stop failures and repairs of nodes and channels.* □

We measure the time complexity of convergence to S in terms of rounds [5]. A round is a minimal, nonempty sequence of protocol steps wherein for each node there exists a step where the node either executes an action or has no actions enabled before or after the step.

Theorem 2 *(Convergence) Starting from any state where T holds, every computation of RST reaches a state where S holds within $2N-3$ rounds, where N is the number of up nodes.* □

(We refer the reader to [4] for formal proofs of all lemmas and theorems in this paper.)

4 Maintaining A Ring

In this section, we employ the second of the two design methods discussed in Section 1 to derive a nonmasking fault-tolerant protocol for maintaining a unidirectional ring. Recall that in this method, we ensure that regardless of the starting state of the protocol, subsequent computation of the protocol converges within a bounded number of steps to a state in which the desired structure, in this case a unidirectional ring, is established.

To motivate our protocol derivation, we first give a constructive proof of

Lemma 4 *A unidirectional ring embedding of length at most $2(N-1)$ and dilation at most 3 can be embedded in an arbitrary but connected graph.*

Proof: Given any connected graph, a rooted spanning tree can be embedded in it. We show by structural induction on the height h of the rooted spanning tree, that a unidirectional ring can be embedded in the rooted spanning tree such that the following two properties are satisfied.

1. In the ring embedding, the successor of the root is a child of the root, and the root is the successor of a grandchild of the root.
2. The length of the ring is at most $2(N-1)$ and the dilation of the ring is at most 3.

Base Case ($h = 0$) : The tree consists of a single node, hence a self-loop on the node is a unidirectional ring embedding that satisfies the properties 1-2. (To verify property 1, we let a leaf node be its own child and, hence, its own grandchild.)

Induction Step ($h > 0$) : From the induction hypothesis, for each child c of the root r of the spanning tree, a ring $ring.c$ can be embedded in the subtree rooted at c such that the properties 1-2 are satisfied. From property 1, there exists a node $fc.c$ that is adjacent to c in $ring.c$ and at a distance of ≤ 1 from c. We construct next a ring embedding in the tree rooted at r from the ring embeddings in the subtrees rooted at the children of r. (See Figure 3.)

Consider the children of r in some fixed total order that starts with $fc.r$. (i) Connect r to $fc.r$. (ii) Invert the successor relation of $ring.c$ for each child c of r. (iii) Connect $fc.c$ of the last child c in the chosen order to r. (iv) For each child c of r, delete from $ring.c$ the edge from $fc.c$ to c, and connect $fc.c$ to the next

sibling of c in the chosen order, provided c is not the last child of r. The resulting structure is a ring that satisfies the properties 1-2. □

In accordance with the existence proof above, we propose to derive a protocol that consists of two layers: The first layer of our protocol maintains, in a nonmasking fault-tolerant fashion, a rooted spanning tree. The second layer of our protocol uses this rooted spanning tree to maintain, in a stabilizing fault-tolerant fashion, the desired unidirectional ring.

We may implement the first layer in terms of any nonmasking fault-tolerant protocol for maintaining a rooted spanning tree, e.g., Protocol *RST* of Section 3. For our purposes, we merely assume that the following variables describing the rooted spanning tree are available to the second layer of each node j.

- $par.j$, denoting the parent node of j ($par.j = j$ if j is the root node).
- $fc.j$, denoting respectively the first node in some fixed total ordering of the children node of j ($fc.j = j$ if j is a leaf).
- $ns.j$, denoting respectively the next sibling of j in the fixed total ordering of the children node of $par.j$ ($ns.j = j$ if j is the last child of $par.j$).
- $even.j$, a boolean denoting whether j is at an even distance from the root node. (To see that $even.j$ is implemented without adding to the time complexity of the first layer protocol, observe that its actions can be augmented to maintain a variable $even.j$. The even value of j is true, if j is a root node, and is the negation of the even value of $par.j$, if j is not a root node.)

We now describe the second layer. Let us represent the unidirectional ring in terms of a "successor" relation; the second layer in each node j maintains the index of the current successor of j in the ring. Following step (i) of the existence proof above, we choose the successor for the root r of the spanning tree to be $fc.r$. Following step (ii) of the existence proof above, we choose the successor for the non-root nodes j of the spanning tree, as follows. Since the successor relation of the ring embeddings of all subtrees is inductively inverted in step (ii), we conclude that the final value of the successor of j depends solely on whether the depth of j in the spanning tree is odd or even: if j is at an odd depth, the successor of j is determined by steps (iii) or (iv); and if j is at an even depth, the successor of j is determined by the inverse of steps (iii) or (iv).

More specifically, given $even.j$, we uniquely determine the successor $s.j$ for each node j in one step, as follows:

(a) j is at odd depth and $ns.j = j$: from step (iii), $s.(fc.j) = par.j$.

(b) j is at even depth and $ns.j = j$: from step (iii) and the inversion argument made above, $s.(par.j) = fc.j$.

(c) j is at odd depth and $ns.j \neq j$: from step (iv), $s.(fc.j) = ns.j$.

(d) j is at even depth and $ns.j \neq j$: from step (iv) and the inversion argument made above, $s.(ns.j) = fc.j$.

Our stabilizing fault-tolerant protocol, *UR*, for the second layer is therefore:

```
node       j (j : 1 .. M)
var        par.j, fc.j, ns.j : 1..M;
           even.j : Boolean;
           s.j : 1..M;

begin
   even.j     ∧ par.j=j    ∧ s.j≠fc.j
      ⟶      s.j := fc.j
[] ¬even.j    ∧ ns.j=j     ∧ s.(fc.j)≠par.j
      ⟶      s.(fc.j) := par.j
[] even.j     ∧ ns.j=j     ∧ s.(par.j)≠fc.j
      ⟶      s.(par.j) := fc.j
[] ¬even.j    ∧ ns.j≠j     ∧ s.(fc.j)≠ns.j
      ⟶      s.(fc.j) := ns.j
[] even.j     ∧ ns.j≠j     ∧ s.(ns.j)≠fc.j
      ⟶      s.(ns.j) := fc.j
end
```

Figure 4 : Protocol *UR*

We illustrate Protocol *UR* with the example of Figure 2. Let us order the children of nodes spatially from left to right. In Figure 2(a), the only child of the root node, 4, is node 2; hence $s.4 = 2$ by the first action. The children of node 2 are 1 and 3, node 1 is childless, and $\neg even.2$ holds; hence $s.1 = 5$ by the second action. Node 3 satisfies $even.3$ and has no sibling to the right; hence, $s.2 = 3$ by the third action. Finally, node 1 is childless and node 3 is its next sibling; hence $s.3 = 1$ by the fifth action. Thus, Protocol *UR* yields the unidirectional ring $(4, 2, 3, 1)$. Likewise, in Figure

2(e), $s.5 = 3, s.3 = 2, s.2 = 1$, and $s.1 = 5$ and , thus, Protocol UR yields the unidirectional ring $(5, 3, 2, 1)$.

In the rest of this section, we show that UR is nonmasking fault-tolerant of fail-stop failures and repairs. We begin by characterizing the invariant predicate S and fault-span predicate T.

Let S be the predicate denoting all states where no action of protocol UR is enabled, and let T be the predicate *true*. Note that, by definition, S is a fixpoint of UR and T is closed in UR as well as fail-stop failures and repairs.

Lemma 5 *At each state of UR where S holds, for each node j, there is some node k (possibly identical to j) such that* $s.j = k$.

Lemma 6 *At each state of UR where S holds, for each up node j, there is some up node k (possibly identical to j) such that* $s.k = j$.

Lemmas 5 and 6 imply that at each state where S holds the in-degree and out-degree of each node j in the graph of the successor relation is 1. Hence, the graph of the successor relation is a collection of node disjoint rings.

Lemma 7 *At each state of UR where S holds, the graph of the successor relation is a ring.*

Observe that each action of UR assigns a successor that is at a distance of at most 3. Further, since the graph of the successor relation is a ring that traverses a rooted spanning tree maintained by the actions of RST, the length of that ring is at most $2(N-1)$.

Theorem 3 *(Convergence): Starting from any fixpoint state of the first layer, every computation of UR reaches a state where S holds within 1 round.* □

5 Related Work, Conclusions

Distributed algorithms for spanning tree construction have been studied extensively. Gallagher et al [6] have presented an elegant solution but their solution is fault-intolerant, i.e., their solution does not solve the spanning tree reconfiguration problem. More recently, stabilizing algorithms for spanning tree reconfiguration have been presented [2, 5, 7] that tolerate failures as well as repairs of both nodes and edges, but these programs are significantly more complex than RST, since they allow for the formation of transient cycles in the graph of the parent variables. Moreover, in the average case, these programs converge much slower than RST. Other programs we are aware of are at best tolerant only to the failures of edges or only the failures and repairs of nodes.

Distributed algorithms for ring embeddings have been presented previously [9, 10], but unlike our algorithm, they do not tolerate fail-stop failures or repairs. Hélary and Raynal [9] present an algorithm that enumerates ring nodes, one at a time, in the order of a depth first traversal of a tree. While this limits the length of the embedded ring to $2(N-1)$, it allows to dilation to be linear in N. The message complexity of their algorithm is $O(N)$, as messages contain a list of nodes already traversed. By way of contrast, a message passing version of RST will use messages of only $O(log\ M)$ size. Rosenstiehl et al [10] have presented a self-synchronizing network of finite state automata that execute in unison from a given initial state to embed a ring of dilation at most three in a tree. Again, the ring is formed one node at a time, as opposed to the constant time ring formation of our approach.

Observe that algorithm RST can also be viewed as an nonmasking fault-tolerant solution to the leader election problem.

We note that our proofs of convergence do not depend on the assumption of fairness for execution of continuously enabled actions. We have thus far assumed fairness only to simplify the exposition. Moreover, there exist refinements of RST and UR that yield read/write atomicity protocols that are nonmasking fault-tolerant as well. We conjecture that the resulting read/write atomicity protocols can be nicely translated into nonmasking fault-tolerant message passing protocols.

References

[1] A. Arora and M. G. Gouda, "Closure and convergence: A foundation of fault-tolerant computing", *IEEE Transactions on Software Engineering* 19(11) (1993), pp. 1015–1027.

[2] A. Arora, "A foundation of fault-tolerant computing", *Ph.D. Dissertation*, The University of Texas at Austin (1992).

[3] A. Arora, M. G. Gouda, and G. Varghese, "Constraint satisfaction as a basis for designing nonmasking fault-tolerance", *Journal of High Speed Networks*, (1994 to appear); *Proceedings of the 14th International Conference on Distributed Computer Systems* (1994), pp. 424–431.

[4] A. Arora and A. Singhai, "Fault-tolerant reconfiguration of trees and rings in networks", *OSU Technical Report CISRC-TR09-1994*.

[5] A. Arora and M. G. Gouda, "Distributed reset", *IEEE Transactions on Computers* 43(9) (1994).

[6] R. G. Gallagher, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees", *ACM Transactions on Programming Lang. and Sys.* 5(1) (1983), pp. 66–77.

[7] G. Varghese, "Self-stabilization by local checking and correction", *Ph.D. Dissertation*, Massachusetts Institute of Technology (1992).

[8] R. Perlman, "An algorithm for distributed computation of a spanning tree in an extended LAN", *Ninth ACM Data Communications Symposium*, Vol. 20, No. 7 (1985), pp. 44–52.

[9] J.-M. Hélary and M. Raynal, "Virtual ring construction in parallel distributed systems", M. Cosnard (ed.), *Parallel Processing*, Elsevier Science, 1988, pp. 333–345.

[10] P. Rosenstiehl, J. R. Fiksel, and A. Holliger, "Intelligent graphs: networks of finite automata capable of solving graph problems", R. C. Read (ed.), *Graph Theory and Computing*, Academic Press, 1972, pp. 219–265.

[11] F. B. Bastani, I.-L. Yen, and I.-R. Chen, "A class of inherently fault-tolerant distributed programs", *IEEE Transactions on Software Engg.* 14(10) (1988), pp. 1431–1442.

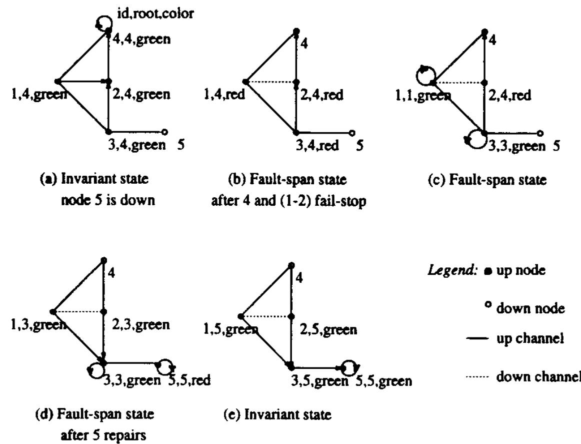

Figure 2

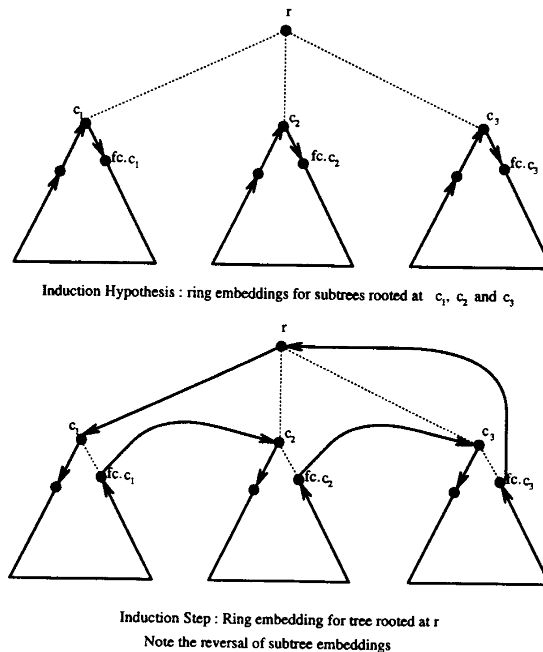

Figure 3