Fault-Local Distributed Mending

(Extended Abstract)

Shay Kutten * David Peleg †

Abstract

As communication networks grow, existing fault handling tools that involve *global* measures such as global time-outs or reset procedures become increasingly unaffordable, since their cost grows with the size of the network. Rather, for a fault handling mechanism to scale to large networks, its cost must depend only on the number of failed nodes (which, thanks to today's technology, grows much slower than the networks). Moreover, it should allow the non-faulty regions of the networks to continue their operation even during the recovery of the faulty parts. This abstract introduces the concepts *fault locality*, and of *fault-locally mendable* problems, which are problems for which there exist correction algorithms (applied after faults) whose cost depends only on the (unknown) number of faults. We show that *any* problem is fault locally mendable. The solution involves a novel technique combining data structures and "local votes" among nodes, that may be of interest in itself.

1 Introduction

1.1 Background

Computer networks and distributed systems are growing very fast. The Internet [6] is estimated to double in size every 6 to 12 months. Currently it connects about 4,000,000 hosts. Most of them belong to end users, whose involvement in control functions is limited (although they do participate in many distributed computing activities). However, at the rate of 100 to 1000 nodes per router, the number of control nodes is estimated in the tens of thousands. Moreover, the internet is not the only huge and fast growing network; working hard to realize the celebrated "information superhighway" are all the main telecommunication companies in the world.

*I.B.M. T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598, USA. *kutten@watson.ibm.com.*

†Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100 Israel. *peleg@wisdom.weizmann.ac.il.* Supported in part by a Walter and Elise Haas Career Development Award and by a grant from the Basic Research Foundation. Part of the work was done while visiting IBM T.J. Watson Research Center.

PODC 95 Ottawa Ontario CA © 1995 ACM 0-89791-710-3/95/08..$3.50

Another characteristic of the emerging networks is their diversity. Numerous manufacturers are involved in producing the hosts, routers and cables for these networks. Even when equipment of the same source is used, there are significant differences in the ages, sizes, speeds, reliability, and many other attributes of the equipment and software used. Moreover, the Internet is not managed by one single authority, but rather by thousands of organizations, governed by very different policies and considerations. Indeed, the parties concerned strive for compatibility, but many differences exist nevertheless.

In such a diverse environment, faults of many different types, leading to information inconsistency, are unavoidable. Indeed, coping with faults, and devising fault-tolerant solutions for various problems, is one of the most active research areas in networking and distributed systems. However, one striking characteristic of this area is that in many of the solutions proposed in the literature, faults are fixed *globally* i.e., by algorithms involving the entire system. Clearly, using global measures to detect and correct errors is becoming more and more unrealistic as networks grow, and it is essential to develop *scalable* fault-handling tools, that is, solutions that can be applied even in large networks. In particular, the cost of such tools is required to grow slower than the system size. It is also required that the non-faulty parts of the networks will be able to continue operating even while the faulty parts are recovering. Otherwise no meaningful work can be done.

The approach we propose for tackling this problem is based on the observation that faults are very often of extremely *local* nature, and involve only a small number of hosts. For example, the famous ARPANET crash [15] was caused by a single node giving all other nodes wrong routing information. Moreover, systems reliability has been increasing, so the number of faults grows much slower than the network sizes. This indicates that the ineffectiveness of global fault handling methods, and the growing need for scalable methods, can potentially be met by the development of what we term *fault-local* tools, whose associated cost does not grow with the size of the networks, but rather only with the extent of failure. (Yet these tools should not assume any knowledge about the extent of the faults.) The concept of fault-locality, and the development of a systematic methodology for designing fault-local error-handling algorithms, are the subject of this paper.

1.2 Problem Description

Let us first describe the types of faults dealt with in this paper. The global state of a distributed system can be represented as a vector whose components are the local states of the various nodes of the system. A transient fault in one node may change the node's state to some other local state that still looks legal. However the global state may no longer be legal. Intensive research efforts were invested in the problem of dealing with this situation, and bringing the system to a correct legal state. A common methodology is based on separating the task into two distinct subtasks, *detection* and *correction*.

In this paper we concentrate on the development of fault-local variants for the correction phase. For the detection part we refer the reader to [7, 8, 9], which proposed the notion of *unreliable fault detectors*. These detectors detect every fault, but may have also "false alarms". Our correction algorithm can, for instance, be started by such detectors, and will not cause any damage in the case of a false alarm. Other relevant work on fault detection was done by [4, 3], suggesting the notion of local detection of faults by local (hence fast) checking.

We initiate the study of fault-local correction algorithms through studying the following basic problem. Consider a problem X on graphs, whose solutions are representable as a function $\mathcal{F}$ on the vertices of the network. The set of legal solutions of X on a given graph G is denoted by $X(G)$.

Consider a distributed network, whose nodes collectively store the representation of some solution $\mathcal{F} \in X(G)$ of the problem X on the graph. Suppose that at time t_0, the memory contents stored at some subset F of the nodes of the network are distorted due to some transient failures. As a result, while the stored values still look locally legal, the representation of the function stored at the network has changed into some inconsistent function $\tilde{\mathcal{F}}$ that is no longer valid.

It is clear that, assuming the problem X is computable, then investing sufficient computational efforts it is possible to *mend* the function, namely, change the values at some of (or all) the nodes, and reconstruct a valid representation of a (possibly different) solution of the same type, $\mathcal{F}' \in X(G)$. The question raised in the current paper is the following:

> Is it possible to distributedly mend the function in time complexity dependent on the number of *failed nodes*, rather than on the size of the entire network?

We term this operation (if and when possible) *fault-local mending*.

We say that the problem X is *fault locally f-mendable* if, following the occurrence of faults in a set F of nodes, the solution can be mended in $O(f(|F|))$ time. When no confusion arises we may say simply *locally f-mendable*, or even just *f-mendable*. A problem X is *fault-locally mendable* if it is fault-locally f-mendable for some function f.

On the face of it, it is far from clear that locally mendable problems exist. It may be even more surprising if such *global* problems exist. After all, if the input of every node may influence the value of the function (as is the case in global functions), how can the function be mended locally to the faulty area? Yet, in this paper we show that *any* problem is fault locally mendable. The solution involves a novel technique combining data structures and "local votes" among nodes, that may be of interest in itself. Intuitively, the faulty values are corrected by a vote in which there is a majority of non-faulty nodes. However, since we hold the votes in small localities (to obtain fault locality in the time complexity) faulty nodes have great opportunities to bias the vote. (We demonstrate this phenomenon in Example 3.1. in the sequel.) The new technique eliminates this bias.

We should remark that there are two inherent limitations to the fault-local approach. First, it should be clear that the precise value of the original function $\mathcal{F}$ may not be recoverable, since we do not necessarily know which nodes suffered faults, or even how many faults occurred. In fact, it is possible for the faults to change the state of each node to look precisely as if some other legal solution $\mathcal{F}'$ is stored in the network, in which case no problem will ever be detected! Another problematic case occurs if the number of faults is larger than $n/2$, in which case it may not be possible to recover the function to its original value. (It is possible if one can rerun the original application that generated the value, but this may not always be possible.) Note that this limitation applies to any kind of correction, not only fault local mending.

While this is indeed a limitation, it may not pose a difficulty in most cases.

Example 1.1 *Consider, for example, the problem of symmetry breaking (or* leader election*), where a legal global state is one in which there is exactly one node in a special local state (*leader*), while all the others are in another state (*not-leader*). It is often not important which of the nodes is this leader, so long as there is a unique leader. Thus, if a set of faults changes a global state in which some node v is the leader, to some other global state where some other node is the leader, there is no need for the mending algorithm to do anything. As explained above, it may in fact not be able to do much, since the new state is legal. Moreover, given just the new global state (and not knowing whether any faults occurred), it may, in fact, be the case that this global state was also the original global state.*

The example above demonstrates also a curious property implied by the definition of fault locality. Consider a locally f-mendable problem X. Suppose that we started from a legal solution $\mathcal{F}$, and through failures at a set of nodes F reached an illegal function $\mathcal{F}'$. As discussed above, there is no way for the mending algorithm to know which nodes suffered faults, or what the original solution was. It follows that the mending algorithm is forced to mend $\mathcal{F}'$ to a legal solution $\mathcal{F}''$ *close* to $\mathcal{F}'$, otherwise it might be too costly if $|F|$ were small. This may in fact be cheaper in some cases than returning to the original $\mathcal{F}$.

Put more formally, let $d(\mathcal{F}, \mathcal{F}')$ denote the distance between the two solutions $\mathcal{F}$ and $\mathcal{F}'$, namely, the number of nodes on which they disagree:

$$d(\mathcal{F}, \mathcal{F}') = |\{v \mid \mathcal{F}(v) \neq \mathcal{F}'(v)\}|.$$

For a problem X, a graph G and a function $\mathcal{F}$, let $cost(\mathcal{F}, X, G)$ denote the distance of $\mathcal{F}$ from $X(G)$,

$$cost(\mathcal{F}, X, G) = \min\{d(\mathcal{F}, \mathcal{F}') \mid \mathcal{F}' \in X(G)\}.$$

Clearly, if $\mathcal{F}'$ is obtained from $\mathcal{F} \in X(G)$ through failures in a set of nodes F, then $cost(\mathcal{F}', X, G)$ is bounded from above by $|F|$. Our observation implies that the complexity of mending a given faulty solution $\mathcal{F}'$ of X should be $O(f(cost(\mathcal{F}', X, G)))$.

While viewed here as a limitation, the notion of closeness was actually promoted in [DH95] as a goal of an algorithm

that adjusts to topological changes (and faults); the algorithm presented there to achieve that, however, performs a global computation even for one change, and is thus *not* fault local. Still, the goal of closeness promoted in [DH95] is intuitively very appealing (even though it did not lead to better complexity); it would be interesting to find whether its benefit can be formally quantified.

The second inherent limitation of fault-local mending has to do with the specific representation of the solution function $\mathcal{F}$ in the network. A crucial observation is that if the function is represented *minimally*, then fault-local mending may be impossible. Consider again the example of leader election. Its minimal representation seems to be keeping a single bit at each node, as described in Example 1.1. However, if a single fault changes the value at the leader to be *not-leader*, then the resulting global state is symmetric, and mending it is as hard as solving leader election from scratch.

Hence any solution approach must be based on making use of additional data structures at the various nodes, typically storing information about the local states of their neighbors. It should be clear that the use of such data structures does not by itself suffice to solve the problem. In fact, it may increase its complexity, since any additional data stored at the nodes of the system is just as prone to erasure or distortion due to faults as is the basic data.

This second limitation does not manifest itself in the persistent bit problem that is dealt with in the greater part of this paper. (In this problem every node needs to keep only one bit.) However, it is very noticeable in other problems (and in the general problem) discussed in Section 7.

In the full paper we will discuss also the difficulties involved in generalizing our solution to weaker models (such as asynchronous model, self stabilization, etc). Some initial steps in this direction have been taken in [KP95b].

1.3 Contributions

In this paper we introduce the notion of fault locality, together with tools to make algorithms fault local, such as *non-minimal function representation* and *restrained voting*.

We further examine the existence of locally mendable functions. Clearly, if the time needed to compute a function from scratch is constant (e.g., some versions of coloring [NS93]) then it is trivially locally mendable. Here, we show that using data structures storing a sufficiently redundant representation, every problem X is locally mendable. (Specifically, every problem X is $c|F|\log|F|$-locally mendable for a small constant c.)[1]

This is done by first introducing a novel and simple algorithm for mending the solution to the global common bit problem. In this problem, initially the network stores a bit in a replicated way. That is, all nodes store an identical bit (i.e., each node v has a variable $\mathcal{M}_v$, and initially all bits are set to the same value). As the result of a failure, the values at some of the nodes have flipped. The problem is to restore a legal situation (of all nodes storing a common bit). We call this mending problem the *persistent bit* problem.

Finally we show how to mend a general function. Of course, a general function is just a collection of bits. However, one cannot just fix each bit separately, since the result may not be a correct solution. For example, assume that the function is the largest node identity in ithe network. If each bit is just corrected separately, then the resulting value may not in fact be the identity of any node. For a worse example consider the Minimum Spanning Tree function. Individual links may not even form a spanning tree. We show how to use the solution for the persistent bit problem in order to mend a general function.

[1] In [12] we show that certain problems, particularly MIS, are $f(|F|)$-locally mendable for f sublinear in $|F|$.

1.4 Related Work

The problem of distributed correction has been studied intensively for many years and in different models (including the one dealt with here). In [10] the notion of a global reset was suggested, by which faults are corrected by a global process that "erases" all the results computed before the faults, and restart a global algorithm that recomputes the results. His algorithms were improved many times, see e.g. [1, 2, 5]. Algorithms using this method are certainly not fault local.

The global reset approach to error correction has two major shortcomings. First, it means that even one fault can cause a global computation, that may take a long time to output a legal result.

Secondly, the global reset method is mainly suitable for a distributed system that cannot (or is not required to) produce "useful work" when some of its nodes suffer a transient fault. In many typical cases, however, it seems desirable for the distributed system to correct itself as locally as possible, letting undamaged regions of the system operate as usual in the meantime.

These two disadvantages of the global correction approach provide the motivation for introducing the notion of fault-locality, which, to the best of our knowledge, has not been discussed before.

Of course, global reset is not the only approach in the literature, or in practice. Its main novelty was its universality: it could be used to mend any problem. An earlier universal approach that may still be the most common in practice is sometimes called *update* (e.g. topology update). The idea is to distribute a notification of a change (in a model where changes are detectable) to all nodes; thus each node can use the information it stores locally to compute the new value of the function. The most famous example of this approach may be computing routing functions (see e.g [15]). This, nevertheless, causes a global computation.

The complexity yielded by the >>> D:Which ??? <<< approach is better than that of global reset for a small number of faults; however, the reset approach was presented in order to bound the worst case. In [2] it was shown that good worst case behaviour can be achieved even in the update approach, while retaining (and even improving) the better complexity for a small number of faults. Others promoted the avoidance of a global reset by taking advantage of the special properties of the function to be mended [2] or by assuming constraints on the faults, such as the assumption that only one fault occurred.

A notion of *local correction* was suggested in [3] in the context of self stabilization. The main meaning of locality there is that each node can act locally to correct a state of an algorithm. However, if the corrected algorithm is global, then the function computed by the corrected algorithm it can be output only after $O(n)$ (number of nodes) or $O(Diameter)$ (diameter of the network). (In fact the example used in in [3] for the corrected algorithm is the global reset algorithm.) One can use the method of [3] in order

to stabilize our algorithm, however, one first needs an algorithm such as ours, since using the method of [3] to stabilize a non-fault-local algorithm (e.g., the global reset algorithm stabilized in [3]) does not yield a fault local algorithm.

1.5 Paper Structure

A formal model is introduced in Section 2. Section 3 highlights the need of the solution to utilize auxiliary data structures, in which nodes store information about other nodes, and outlines some of the difficulties involved.

Section 4 provides an overview of our solution. Section 5 presents our mending algorithm for the global persistent bit problem. Correctness proof and analysis are given in Section 6. Section 7 describes how to use the global persistent bit problem in order to fault-locally mend any function.

2 Model and Definitions

We model a distributed system as a graph $G = (V, E)$ where V is the set of the system (or network) nodes, $|V| = n$, and E is the set of links. The nodes do not share memory, and the only way for two nodes to communicate is by sending a message over a link connecting them if such a link exists. Our solution deals with a synchronous network, that is, the computation proceeds in rounds such that in every round each node receives all the messages sent to it by its neighbors in the previous round, computes and possibly sends messages to one or more of its neighbors over the links connecting it to these neighbors. Moreover, we assume that all faults occur simultaneously at time t_0. We adopt the model employed in previous studies of locality issues [11, 14], in which arbitrary size messages can be transmitted in a single time unit. (Our solution for a single persistent bit, in fact, does not require very large messages. Large messages are required, however, in our solution for a general function, for certain functions.)

We need a few definitions concerning the immediate surroundings of sets of nodes. Let $\Gamma_i(v)$ be the set of all nodes whose distance from v in G is less than or equal to i. For a set of nodes W, let $\Gamma_i(W) = \bigcup_{w \in W} \Gamma_i(w)$.

We first consider the problem of mending a common bit on a given graph G. The common bit is represented by a vector of bits $\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_n})$, which are all the same (i.e., the vector $\mathcal{M}$ is either $\bar{0}$ or $\bar{1}$.) The representation $\mathcal{M}$ is stored distributedly, with each node v storing its own bit $\mathcal{M}_v$. As the result of a failure, the values at $|F|$ of the nodes have flipped. The problem is to restore a legal situation (of all nodes storing a common bit).

Later on, we consider any problem X. In this case, the solution $\mathcal{F}$ is represented by a more general structure $\mathcal{F}(v)$ at every node v in the network. Still, for simplicity we shall formulate the definitions in the remainder of this section for the simple case of the common bit problem.

Our approach allows a more general representation, in which in addition to the output bit $\mathcal{M}_v$, each node v may maintain an additional data structure $\mathcal{D}_v$.

Let $\mathcal{M}_v^*$ be the output (namely, the representation of the common bit) before the faults, and let $\mathcal{M}_v$ be the current representation of the common bit (when the mending operation may be required). Likewise, let $\mathcal{D}_v^*$ be the data structure stored at v before the faults, and let $\mathcal{D}_v$ be the current structure stored at v. Thus the bits $\mathcal{M}_v$ and the structures $\mathcal{D}_v$ are the input for the mending task, and both of them must be mended.

Formally, the set of faulty nodes is defined as $F = \{v \mid \mathcal{M}_v \neq \mathcal{M}_v^* \text{ or } \mathcal{D}_v \neq \mathcal{D}_v^*\}$. Likewise, the healthy nodes are $H = \{v \mid \mathcal{M}_v = \mathcal{M}_v^* \text{ and } \mathcal{D}_v = \mathcal{D}_v^*\}$ (Clearly, $\mathcal{M}_v^*$ and $\mathcal{D}_v^*$ are not known to v.)

Definition 2.1 *Given the vector $\mathcal{M}$, the* conflict set *of a node v to distance ℓ, denoted $\mathcal{C}_\ell(v, \mathcal{M})$, is the set of nodes in v's ℓ-neighborhood with a conflicting value, i.e.,*

$$\mathcal{C}_\ell(v, \mathcal{M}) = \{u \in \Gamma_\ell(v) \mid \mathcal{M}_u \neq \mathcal{M}_v\}.$$

We remark that although the general formulation allows the use of auxiliary data structures $\mathcal{D}$, our solution for the global persistent bit problem makes use of no additional structures beyond the bit itself[2].

3 The need for supporting data structures

One preliminary issue that needs to be discussed has to do with the representation of the function in the network. A first naive solution one may suggest for the problem of mending a corrupted function is to have the nodes in the conflict set $\mathcal{C}(\mathcal{M})$ (namely, the nodes that directly observe an inconsistency with respect to their neighbors) run some resolution protocol among themselves, without involving the rest of the nodes of the graph.

Assume for a moment that it is possible to efficiently correct $\mathcal{M}$ in such a way[3]. Unfortunately, this alone cannot guarantee that the complexity of the mending process is bounded as a function of the number of corrupted nodes, since the conflict set might be much larger than the set of corrupted nodes.

To demonstrate this problem, recall Example 1.1 in a complete graph. Suppose that a single node v (the leader) has failed in such a way that its local representation of the function disagrees with the rest of the nodes (it is no longer the leader). This results in an assignment $\mathcal{M}$ whose conflict set consists of the entire graph. If nodes have unique identities then the leader can be recomputed from scratch in constant time in a complete graph. However, if nodes do not have unique identities (and a randomized solution is required) it is not clear that this can be done.

The general approach we adopt in order to overcome this problem is based on using some auxiliary data structures, in addition to the bits $\mathcal{M}_v$. Such data structures should enable each node to store information about its state at other nodes in its close vicinity. Ideally, this idea should enable a corrupted node to recover its original state by consulting the records of neighboring nodes. The representation of the function at hand will be considered as consisting of all data structure it uses (as indeed defined in the Section 2), and the definition of mending will be with respect to the entire representation. (Note that the global persistent bit problem is somewhat exceptional in this respect, in that additional data structures are not needed. This is because the very nature of the problem implies that a single bit can simultaneously capture the value of the represented function at both the node v itself and all other nodes!)

[2] unlike our solution for general problems, or for MIS in [12]

[3] It may be possible to solve the persistent bit problem; however, in a forthcoming paper [12] we show an example where the problem of checking whether this can be done is NP-Complete

Given the use of auxiliary data structures for mending, it is easy to overcome the difficulty discussed above concerning the complete graph. A simple correction using supporting data structures would be: Let each node remember the value $\mathcal{M}_w$ of the neighboring nodes w. Now the value of a corrupted node can be corrected in a straightforward way in $O(1)$ time by consulting its neighbors.

Indeed, our subsequent solutions make extensive use of such supporting data structures. (As discussed earlier, this is done in a degenerate form for the global persistent bit problem, but more explicitly later, when extending the solution to any problem). Yet this alone does not solve the entire problem. Before introducing our solution let us illustrate the difficulties involved. This can help understand the intuition behind our algorithm.

The simple utilization of data structures as in the above example runs into the obvious difficulty, that the data stored at a node for its neighbors might get corrupted just as easily as its own data. Thus a node inspecting its neighbors' data structure might encounter conflicting views regarding its data.

A natural idea for attacking this difficulty would be to require a node in conflict to consult *all* its neighbors, and adopt the *majority* of their views as its own. Returning to our complete graph example, so long as fewer than $n/2$ of the nodes are corrupted, the corrupted nodes can be corrected by using a majority vote. Of course, once the number of corrupted nodes exceeds $n/2$, global mending is necessary, but in this case the extra cost is justified by the size of F (or in other words, the complexity of the mending operation is still well bounded as a function of $|F|$).

Unfortunately, unlike the situation in the complete graph, there are many other topologies in which a small coalition of corrupted nodes can control the majority of neighbors around many nodes, and thus cause the need for extensive mending. In particular, the following simple example shows that just a few faulty nodes may cause an erroneous result of the majority voting in many nodes.

Example 3.1 *Consider a graph $G = (V, E)$ where $V = \{a, b\} \cup \{1, \ldots, n-2\}$, and each $1 \leq i \leq n-2$ is connected to both a and b, but not to any other node (see Fig. 1). If a and b alone are faulty, consulting the data structures stored at neighbors will suffice to sway all of the nodes into erroneously modifying their function representation.*

Let us comment that the results of [13] imply that it is impossible for a small coalition to control *all* the neighborhood majorities in the entire graph. Unfortunately, the results of [13] do not exclude the possibility of a very small coalition controlling *most* neighborhoods, as illustrated in the above example, which is devastating enough in our case.

4 Overview of Our Solution for the Persistent Bit Problem

Let us first describe a simplified version of the algorithm for the persistent bit problem, based on the assumption that the number of faults $|F|$ is known in advance. (This assumption is made merely for ease of explanation; we get rid of it later.) Then the output of each node v is determined by a *majority vote* among its neighbors to distance $2|F|$ (including itself), based on their stored value. If the result of this polling is different than $\mathcal{M}_v$ then v changes its $\mathcal{M}_v$ accordingly. Note that following this poll the original correct value of $\mathcal{M}_v^*$ for any v is recovered.

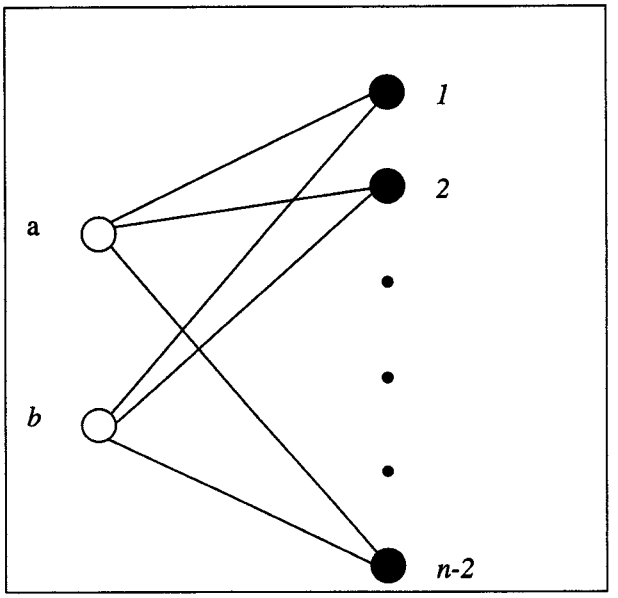

Figure 1: A small set $\{a, b\}$ controlling the majority of neighbors of all other nodes. Initial MIS contains $\{1, \ldots, n-2\}$. Failure of a and b alone forces conflicts in all nodes.

Let us now turn to the general case, where it is not assumed that $|F|$ is known. In this case, our algorithm runs in phases, performing a search on the number of faults $|F|$. In phase i it is "guessed" that the number $|F|$ of faults is not larger than 2^i.

Several things can go wrong when the guess is too small. The most severe problem is that the computations carried out in the first phases might increase the number of "currently faulty" nodes. That is, denoting by F_i the set of v nodes whose bit value $\mathcal{M}_v$ at the beginning of the ith phase is different from their original value $\mathcal{M}_v^*$, while $F_1 = F$, the set F_2 is affected by the outcome of phase 1. In particular, the assignment of new bit values to some of the nodes in phase 1 might cause many nodes whose original value was correct to erroneously change their value to an incorrect one, thus effectively making the set F_2 much larger than F, and so on.

The only way to completely wipe out the influence of the previous phase would be to make all participating nodes return to their initial values. Indeed, this re-initialization operation is applied at nodes that decide at the end of a phase to participate in subsequent phases. (For that, the initial value of $\mathcal{M}_v$ is saved in a variable $\mathcal{M}_v^{sav}$.) But applying such a re-initialization operation globally might be highly time consuming, since it may very likely happen that the assignment in the neighborhood of some nodes in the conflict set C_i is consistent at the end of the phase, and hence these nodes are ready to terminate, while some other nodes observe inconsistencies, implying the necessity of another phase. Therefore re-initialization of all nodes requires broadcast of a message from the latter nodes to the former, which might be time consuming.

It is therefore necessary to ensure two things: first, that the algorithm is able to detect that the guess was wrong, and secondly, that the operations of the algorithm under the

For every node v, save $\mathcal{M}_v^{sav} = \mathcal{M}_v$.
For $i = 1$ to $\log n$ **do**

1. For every node v, let $\gamma_i(v) = |\Gamma_{2^i}(v)|$.
2. Let $\texttt{Non\_Voters}_i = \{v \mid |\mathcal{C}_{2^i}(v, \mathcal{M}^{sav})| > 2^i\}$ and $\texttt{Voters}_i(v) = \Gamma_{2^i}(v) \setminus \texttt{Non\_Voters}_i$.
3. Every node v with $\mathcal{C}_1(v, \mathcal{M}) \neq \emptyset$ broadcasts a message requesting a "poll" to distance 2^i.
4. Every node v receiving a "poll" message performs a poll among its set of voters to distance 2^i. The outcome is determined as follows:
   (a) **Let** $\mathcal{M}_v \leftarrow 0$ if $|\{v \in \texttt{Voters}_i(v) \mid \mathcal{M}_w^{sav}(v) = 0\}| > \frac{\gamma_i(v)}{2}$.
   (b) **Let** $\mathcal{M}_v \leftarrow 1$ if $|\{v \in \texttt{Voters}_i(v) \mid \mathcal{M}_w^{sav}(v) = 1\}| > \frac{\gamma_i(v)}{2}$.
   (c) **If** neither of the two votes has the majority /* due to non-voters */ then leave $\mathcal{M}_v$ unchanged.

Figure 2: Algorithm COMMON_CORRECT.

wrong guess did not cause "too much damage". We achieve this behavior by attempting to prevent seemingly faulty nodes from participating in the voting process and gaining undue influence. Towards that end, we introduce one additional "screening" step before performing our polling. The idea is that nodes with a suspiciously high number of conflicts with neighbors (where "high" here is relative to the phase) will be "guessed" faulty, and consequently will be barred from participating in the voting. It is important to note that despite this possibility of non-voters, the votes carried at the nodes will still require a strict majority, namely, more than half of the entire 2^i-neighborhood (not just half of the voters).

5 Mending algorithm

We are now ready to describe our mending algorithm for the common bit problem. The algorithm, COMMON_CORRECT, is given below in Figure 2.

The algorithm starts by saving the input values of $\mathcal{M}$ in variables $\mathcal{M}^{sav}$. These input values will be used from time to time throughout the run of the algorithm (in particular, whenever the algorithm requires a node to consult the data structure, only these values will be used). The variables $\mathcal{M}$ will be used for recording the output of the computation. Specifically, whenever a node v selects a new value for its common bit assignment, it stores it in $\mathcal{M}_v$.

As outlined earlier, the algorithm conducts majority votes to determine the output of some nodes, polling the values of their neighbors. However, the nodes with many collisions (where "many" here is with respect to the phase we're currently in) are "guessed" to be faulty, and thus are not allowed to vote.

In each phase i, the algorithm performs polls on neighborhoods of size 2^i, and also separates nodes with many conflicts from ones with few conflicts using the threshold 2^i.

6 Correctness and analysis

Let us first describe the general structure of the proof. In Lemma 6.5 (using Lemma 6.4) we show that, assuming $|F| \leq n/2$, there is a sufficiently large i such that phase i computes a valid assignment of a common bit, which in fact must equal the original input.

Remark 6.1 *For F larger than $n/2$ the algorithm may fail, in which case the common bit may be computed (upon detecting a local inconsistency still persisting at some node) via a global method, costing time $O(n)$ at the most in the model employed here. This still qualifies as an $O(|F|)$-fault-local mending operation given the size of F. The choice of the specific global resolution procedure to be collectively employed by the vertices of the graph for computing the common bit upon such an event is dependent on the particular problem at hand, namely, the specific application that created the common bit in the first place. For instance, in some cases it may be sufficient to set the bit to some pre-fixed value, whereas in some other cases the common bit may have some specific meaning, dependent on the graph topology or the inputs, in which case it should be computed from scratch. In other cases, where the task is required just to remember some information that cannot be regenerated, this information is lost if the majority of the nodes are faulty.*

An interesting property showing up in the analysis is that the complexity of mending the common bit depends on the size of the set F_i of nodes whose output $\mathcal{M}_v$ at the beginning of phase i is different than the original value $\mathcal{M}_v^*$. Note that these could be either faulty nodes, or healthy nodes that erroneously computed a new $\mathcal{M}_v$ during previous phases. In Lemma 6.2 we show that the number of such nodes F_i in phase i is not much larger than the number of faults F, by showing that the number of nodes in F_{i+1} is not much larger than the number of nodes that participated in the voting in phase i.

Let F_i denote the set of nodes whose $\mathcal{M}$-value at the beginning of phase i is inconsistent with the "correct" one, i.e., $F_i = \{v \mid \mathcal{M}_v \neq \mathcal{M}_v^*\}$.

Also, let Z_i denote the set of nodes that detected a conflict with a neighbor in Step 3 of the algorithm, and let W_i denote the set of nodes that participate in the polling in phase i. Note that

$$W_i = \Gamma_{2^i}(Z_i). \tag{1}$$

Lemma 6.2 *In any phase $k \leq \log n$, the set F_k satisfies $|F_k| \leq \min\{(2k-1)|F|, n\}$.*

Proof: The upper bound of n on F_k is trivial. The upper bound of $(2k-1)|F|$ is proved by induction on k. The claim is clear for the beginning of phase $k = 1$. Now let us assume that the claim holds for phase k and look at the beginning of phase $k+1$.

Nodes enter F_{k+1} either if they are in F_k or as a result of performing a majority vote with an erroneous result. By the inductive hypothesis, the size of the first set is at most $(2k-1)|F|$. The second of those sources is bounded as follows. Let Y denote the set of nodes that got the wrong value as the result of a wrong vote in phase k. Let μ denote the number of pairs (u, v) such that $u \in Y$, $v \in F \setminus \texttt{Non\_Voters}$, $dist(u, v) \leq 2^k$, and $\mathcal{M}_u^{sav} \neq \mathcal{M}_v^{sav}$.

Observe that since a node can enter Y only if it has more than $\gamma_k(v)/2 \geq 2^{k-1}$ nodes from $F \setminus \texttt{Non\_Voters}$ in

its 2^k-neighborhood, $\mu \geq 2^{k-1}|Y|$. On the other hand, $\mu \leq 2^k|F \setminus \texttt{Non\_Voters}_k| \leq 2^k|F|$. Hence $|Y| \leq 2|F|$. ∎

Lemma 6.3 *Let i be such that $2^i \geq |F|$. Then $\texttt{Non\_Voters}_i$ contains only nodes from F.*

Proof: A node $v \in H$ can have conflicts only with nodes of F, hence it can have at most $|F|$ conflicts overall. Thus $\texttt{Non\_Voters}_i$ does not contain it by definition. ∎

Lemma 6.4 *Let i be such that $2^{i-1} > |F|$. Then for every node w that performed a poll, the voted value for $\mathcal{M}_w$ equals $\mathcal{M}^*_w$.*

Proof: For such i, each node w has $\gamma_i(w) \geq 2^i$ neighbors (including itself), out of which at most 2^{i-1} are in F. By the previous lemma, none of its healthy neighbors from H can be in $\texttt{Non\_Voters}_i$. Thus, at least $\gamma_i(v) - 2^{i-1} > \gamma_i(v)/2$ do vote for a value for $\mathcal{M}_w$ which is equal to $\mathcal{M}^*_w$. The lemma follows. ∎

Lemma 6.5 *1. If $|F| < 2^{i-1}/i$ for some $i \leq \log n$, then at the end of phase i, $\mathcal{M}$ is a valid common bit assignment, and moreover, it is equal to $\mathcal{M}^*$.*

2. If $|F| < n/2$, then at the end of phase $\log n$, $\mathcal{M}$ is a valid common bit assignment, and it is equal to $\mathcal{M}^*$.

Proof: The first claim is proved as follows. Suppose that $|F| < 2^{i-1}/i$. By Lemma 6.4 all the nodes that participated in the polling process end with their values from $\mathcal{M}^*$. It remains to argue that every node from F_i must actually participate in the polling in phase i.

Consider a node $v \in F_i$. By Lemma 6.2, v must be within distance $2i|F|$ from some node with a correct value, and therefore within distance $2i|F|$ from some node $z \in Z_i$ who has detected a conflict with a neighbor in Step 3 of the algorithm. By the assumption of the lemma, $2i|F| \leq 2^i$, hence v belongs to the set W_i of nodes that participate in the polling.

For the case of $n/\log n < |F| < n/2$, we use a similar argument, noting that at the last phase, if conflicts still exist then they are visible to all nodes (including specifically all nodes of $F_{\log n}$). ∎

Lemma 6.6 *If $|F| < n/2$, then Algorithm* COMMON_CORRECT *reaches a legal common bit assignment in time $O(\min\{|F|\log|F|, n\})$.*

Proof: Let us first examine the case where $|F| \leq n/\log n$. Let J be the first phase satisfying $2^{J-1}/J > |F|$. By the previous lemma, the algorithm will halt on phase J. The total time consumed by phases 1 through J is bounded by $T = \sum_{j=1}^{J} O(2^j) = O(2^J)$. By the choice of J it follows that $2^{J-2}/(J-1) < |F| \leq 2^{J-1}/J$, and hence $T = O(2^J) = O(|F|\log|F|)$.

Turning to the case where $n/2 > |F| > n/\log n$, we note that the algorithm must terminate after the first iteration J in which the entire graph was reached from every node. This iteration cost time $O(n)$ and all the previous iterations combined cost no more than that as well. ∎

Our analysis establishes the following theorem.

Theorem 6.7 *The common bit problem is locally $(|F|)$-mendable.*

7 Universal Fault Local Mending

In this section we describe how to use the above solution for the global persistent bit problem in order to derive a fault-local mending algorithm for any computable problem X.

Given a problem X whose solutions are functions $\mathcal{F} \in X(G)$, represent the function on G using auxiliary data structures, by storing the entire representation of $\mathcal{F}$ on the whole network, i.e., the vector $\xi = (\mathcal{F}(v_1), \ldots, \mathcal{F}(v_n))$, at every node. Mending is performed by treating each bit of this representation vector ξ separately, and applying to it the common bit algorithm.

Observe that by Claim 2 of Lemma 6.5, if $|F| \leq n/2$ then the mending operation on any bit $\mathcal{M}$ in ξ will bring the output back to the original $\mathcal{M}^*$. Hence as long as $|F| \leq n/2$, the result of the mending operation will be the original representation ξ. The cost of this mending operation is $O(\min\{|F|\log|F|, n\})$ by Lemma 6.6.

In case $|F| > n/2$, this algorithm may fail. If it does fail, however, then an inconsistency must be detected for at least some bit $\mathcal{M}$ of the vector ξ. In this case, any processor detecting an inconsistency broadcasts a message to this effect throughout the entire network. In response to such message, the processors collectively employ a global procedure for computing a new solution $\mathcal{F}'$ for the problem X. This operation requires at most $O(n)$ time in our model (for instance, by collecting all the inputs to one node, computing a solution locally, and broadcasting the solution to all the nodes in the network). By assumption, this complexity is $O(|F|)$.

Theorem 7.1 *Every computable problem X is fault locally mendable.* ∎

References

[1] Yehuda Afek, Baruch Awerbuch, and Eli Gafni. Applying static network protocols to dynamic networks. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, October 1987.

[2] Baruch Awerbuch, Israel Cidon, and Shay Kutten. Optimal maintenance of replicated information. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990.

[3] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proc. 32nd IEEE Symp. on Foundation of Computer Science*, pages 268–277, October 1991.

[4] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self stabilizing protocols for general networks. In *Proc. 4th Int. Workshop on Distributed Algorithms*, (WDAG), Springer-Verlag LNCS, September 1990.

[5] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir and G. Varghese, Time-Optimal Self Stabilizing Synchronization, *Proc. 1993 ACM Symp. on Theory of Computing*, San Diego, California, pp. 652–661, May 1993.

[6] D. E. Comer. Internetworking with TCP/IP. Prantice Hall, New Jersey, US, 1991.

[7] T. D. Chandra and S. Toueg. Unreliable Failure Detector for Asynchronous Systems. In *Proc. 10th ACM Symposium on Principles of Distributed Computing*, Montreal, Canada, August 1991, pp. 325-340.

[8] T. D. Chandra, V. Hadzilacos, and S. Toueg. The Weakest Failure Detector for Solving Concensus. In *Proc. 11th ACM Symposium on Principles of Distributed Computing*, Vancouver, Canada, August 1992, pp. 147-158.

[DH95] S. Dolev and T. Herman. SuperStabilizing Protocols for Dynamic Distributed Systems. In *2nd Workshop on Self-Stabilizing Systems*. Las Vegas, Nevada, May 1995.

[10] Steven G. Finn. Resynch procedures and a fail-safe network protocol. *IEEE Trans. on Communications*, COM-27(6):840–845, June 1979.

[11] A. V. Goldberg, S. Plotkin, and G. Shannon, Parallel symmetry breaking in sparse graphs, *Proc. 19th ACM Symp. on Theory of Computing*, May 1987.

[12] S. Kutten and D. Peleg, Tight Fault-locality, Manuscript, 1995.

[KP95b] S. Kutten and D. Peleg. Fault-Local Self-Stabilization, In preparation.

[9] W. Lo and V. Hadzilacos. Using Failure Detectors to Solve Concensus in Asynchronous Shared Memory Systems. In *Proc. 8th Int. Workshop on Distributed Algorithms*, Terschelling, The Netherlands, September 1994, pp.280-295.

[13] N. Linial, D. Peleg, Y. Rabinovich and M. Saks, Sphere Packing and Local Majorities in Graphs, *Proc. 2nd Israel Symp. on Theory of Computing and Systems*, Natanya, Israel, June 1993, 141–149.

[14] N. Linial, Locality in distributed graph algorithms, *SIAM J. on Comput.*, 21:193–201, 1992.

[15] J. M. McQuillan, I. Richer and E. C. Rosen, The New Routing Algorithm for the ARPANET. *IEEE Trans. on Communications*, COM-28, May 1980, pp. 711-719.

[NS93] M. Naor and L. Stockmeyer, What Can Be Computed Locally? In *Proc. of the 25th ACM Symp. on the Theory of Computing*, San Diego, California, May 1993, pp. 184-193.