SICOMP 27-1 (1998), pp.247-290

# FULLY POLYNOMIAL BYZANTINE AGREEMENT FOR $n > 3t$ PROCESSORS IN $t + 1$ ROUNDS*

JUAN A. GARAY† AND YORAM MOSES‡

**Abstract.** This paper presents a polynomial-time protocol for reaching Byzantine agreement in $t + 1$ rounds whenever $n > 3t$, where $n$ is the number of processors and $t$ is an *a priori* upper bound on the number of failures. This resolves an open problem presented by Pease, Shostak and Lamport in 1980. An early-stopping variant of this protocol is also presented, reaching agreement in a number of rounds that is proportional to the number of processors that actually fail.

**Key words.** Byzantine agreement, consensus, distributed computing, fault tolerance, computer security.

**AMS subject classifications.** 68M10, 68M15, 68Q22, 94C12

## 1. Introduction.

The Byzantine agreement problem (BA), introduced by Pease, Shostak and Lamport in [22], is recognized as a fundamental problem in fault-tolerant distributed computing. Over the last decade or more, the problem has received a great deal of attention in the literature, and has become a testbed for a variety of models for distributed computing (see [15] for an early survey on the subject). While Byzantine agreement has been studied extensively over the years, the original model in which faulty processors can act in arbitrarily malicious ways has continued to withstand a complete analysis. In [22], the authors presented a protocol that solves the problem (then still referred to as the *interactive consistency problem*) in $t+1$ rounds whenever $n > 3t$. Here $n$ is the total number of processors and $t$ is an *a priori* upper bound on the number of faulty processors possible. They also proved that no solution for $n \leq 3t$ exists, while Fischer and Lynch later showed that $t + 1$ rounds were necessary in the worst-case run of any BA protocol [16]. The protocol presented in [22], however, required the processors to send exponentially long messages and perform exponentially many steps of computation. The design of more efficient protocols is presented in [22] as an open problem, and has been the subject of many subsequent papers. This paper presents a BA protocol for $n > 3t$ that halts in $t + 1$ rounds and uses only a polynomial amount of communication and computation.

This work is based on a long sequence of papers whose goal was to reduce the complexity of the protocol while maintaining good performance in terms of the number of rounds necessary for agreement. Polynomial-time BA protocols for $n > 3t$ that halt in more than $2t$ rounds (see, e.g., [11, 24]) have been known as of 1982. In 1985 Coan presented a family of BA protocols for $n > 4t$ that, for every $d$, halt in $t + t/d$ rounds, and require messages of size $O(n^d)$ [8]. However, Coan's protocols require exponential local computation. Bar-Noy, Dolev, Dwork, and Strong later improved on this result, providing protocols with essentially the same round and communication behavior, but requiring only polynomial computation [2]. These protocols thus pro-

* An early version of this work was presented in the 1993 STOC conference [18].

† IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 (garay@watson.ibm.com). Part of this work was done while the author was on leave at Weizmann Institute of Science.

‡ Department of Applied Math and Computer Science, Weizmann Institute of Science, Rehovot, 76100 Israel (yoram@wisdom.weizmann.ac.il). This work was supported in part by a Helen and Milton A. Kimmelman Career Development Chair.

| Protocol | | $n$ | rounds | comm. | comp. |
|---|---|---|---|---|---|
| [PSL] | 80 | $3t+1$ | $t+1$ | $\exp(n)$ | $\exp(n)$ |
| [DFFLS,TPS] | 82 | $3t+1$ | $2t+c$ | $\text{poly}(n)$ | $\text{poly}(n)$ |
| [C1] | 85 | $4t+1$ | $t+\frac{t}{d}$ | $O(n^d)$ | $\exp(n)$ |
| [DRS,BD,C2] | 86 | $\Omega(t^2)$ | $t+1$ | $\text{poly}(n)$ | $\text{poly}(n)$ |
| [BDDS] | 87 | $3t+1$ | $t+\frac{t}{d}$ | $O(n^d)$ | $O(n^d)$ |
| [MW] | 88 | $6t+1$ | $t+1$ | $\text{poly}(n)$ | $\text{poly}(n)$ |
| [BGP1] | 89 | $3t+1$ | $t+\frac{t}{d}$ | $O(c^d)$ | $O(c^d)$ |
| [BG1] | 89 | $4t+1$ | $t+1$ | $\text{poly}(n)$ | $\text{poly}(n)$ |
| [CW] | 90 | $\Omega(t\log t)$ | $t+1$ | $\text{poly}(n)$ | $\text{poly}(n)$ |
| [BG2] | 91 | $(3+\epsilon)t$ | $t+1$ | $\text{poly}(n)\cdot O(2^{\frac{1}{\epsilon}})$ | $\text{poly}(n)\cdot O(2^{\frac{1}{\epsilon}})$ |
| This paper | | $3t+1$ | $t+1$ | $\text{poly}(n)$ | $\text{poly}(n)$ |

TABLE 1
*History of Byzantine agreement (partial list).*

vide a tradeoff between the number of rounds required and the size of messages used, and prove that $2t$ rounds are not necessary for polynomial-time BA protocols. $(t+1)$-round polynomial-time BA protocols for $n = \Omega(t^2)$ were presented in 1986 by Dolev, Reischuk and Strong [13]. In 1988 Moses and Waarts presented the first polynomial $(t+1)$-round protocol with linear resilience: It required only that $n > 8t$, and was later improved to handle $n > 6t$ [21]. In 1989 Berman and Garay presented a polynomial protocol for $n > 4t$ [3], which they improved in 1991 to handle $n > (3+\epsilon)t$ for any $\epsilon > 0$ [4]. At the cost of requiring more processors ($\Omega(t\log t)$), Coan and Welch developed a polynomial protocol that uses one-bit messages and asymptotically optimal total bit transfer [10]. Table 1 presents a summary of these and related results.

Our work starts out using observations and techniques developed in [2, 21, 3]. In particular, we start from the formulation by Bar Noy *et al.* of the exponential protocol for $n > 3t$ in terms of a two-stage process: In the first stage, processors exchange information for $t+1$ rounds, and each processor stores the information it receives in a tree-like data structure. In the second stage, each processor computes *resolved* values for each node in a bottom-up fashion, where the resolved value of a node is a function of the resolved values of its children in the tree. The resolved value of the root of the tree is the value the processor decides on. In order to obtain our result, we need to develop new techniques to handle the great powers that faulty processors have when $n > 3t$. In particular, we study the class of functions that can serve in the second stage of the algorithm for computing resolved values for the nodes. We present a new class of *admissible* resolve functions, that alternate between favoring the value 0 at one level of the tree and favoring 1 at the next level. By extending the methods for fault detection and fault masking used, we are able to present a particular admissible resolve function, which restricts the freedom of faulty processors. Given these methods, this function forces the number of processors that are detected as faulty by all nonfaulty processors to be large in runs in which the processors' trees grow exponential. This in turn makes it possible to apply a monitor voting technique along the lines of the Cloture Votes technique of Berman and Garay [3] to obtain a polynomial BA protocol.

A few comments are in order regarding the related *randomized* Byzantine agreement problem. Rabin showed that a shared coin can be used to attain randomized agreement in a constant number of rounds [23]. Later, Feldman and Micali presented

an algorithm for constructing such a coin, thereby providing an optimal protocol for this problem [17]. In the protocol of [17], however, the Decision property was guaranteed only with probability 1, and the protocol has runs in which no processor ever reaches a decision. Moreover, there is no finite bound $K$ such that the processors that do decide, do so in $K$ rounds. Indeed, there is no obvious way, to transform such a probabilistic protocol into a polynomial protocol in which nonfaulty processors never violate the conditions of Byzantine agreement (Agreement and Validity), and yet they all decide in fewer than $t+1$ rounds with probability 1. A very close approximation of this was obtained by Goldreich and Petrank [19]. They presented a probabilistic Byzantine agreement protocol with a constant expected number of rounds, that is guaranteed to always halt in $t + O(\log t)$. Their protocol is based on a sequential combination of a probabilistic protocol with a deterministic $(t+1)$-round protocol. Before our paper, their scheme when applied to the case of $n > 3t$ would yield a protocol with an exponential worst-case complexity in terms of both computation and communication. Using our protocols in their scheme, it is now possible to obtain such a probabilistic protocol for $n > 3t$ with polynomial-time worst-case complexity. Using different techniques, Zamsky has recently improved to $t+1$ the worst-case running time of [19], while maintaining linear, but non-optimal, resiliency [27].

The remainder of the paper is organized as follows. In Section 2 we describe the model and formally define the problem. In Section 3 we review EIG, the exponential information gathering protocol of Bar-Noy *et al.* In Section 4 we review the techniques that made polynomial-time protocols possible for $n > 4t$. Section 5 is the first step towards a polynomial solution for $n > 3t$. We introduce a general, radically different class of resolve functions. This class of functions is then used in Section 6 as the basis of a protocol for a slightly generalized version of Byzantine agreement. In Section 7 we present the concrete function of our choice, while in Section 8 we present `Sliding-flip`, the polynomial-time protocol for $n > 3t$. Finally, in Section 9 we modify the protocol so as to terminate in $\min\{t+1, f+3\}$ rounds, where $f < t$ is the actual number of failures that occur in the run.

**2. Preliminary Definitions.** For ease of exposition of our protocols we shall be concentrating on the following variant of Byzantine agreement, sometimes also called the *Consensus* problem, which is defined as follows: Given are $n$ processors, at most $t$ of which might be faulty. Each processor $i$ has an initial value $v_i \in \{0,1\}$. Required is a protocol with the following properties:

(i) **Decision:** Every non-faulty processor $i$ eventually irreversibly "decides" on a value $d_i \in \{0,1\}$.

(ii) **Agreement:** The non-faulty processors all decide on the same value.

(iii) **Validity:** If the initial values $v_i$ of all nonfaulty processors are identical, then $d_i = v_i$ for all nonfaulty processors $i$.

In the other common variant of Byzantine agreement there is a distinguished leader with a single initial value $v$. The Agreement and Decision conditions remain the same, while the Validity condition requires that if the leader is nonfaulty, all nonfaulty processors should decide on the leader's value $v$. Protocols for both variants are practically identical, and everything we say here applies to the single leader case with only minor modifications.

Throughout the paper we use $t$ to denote an upper bound on the number of faulty processors. We assume the standard model for Byzantine agreement, in which processors may fail in arbitrarily malicious ways (see, for example, [16]). Each processor can communicate directly with every other processor via a reliable point-to-point

channel. Finally, the processors are synchronous, and their communication proceeds in synchronous rounds: In a round, a processor can send one message to each of the other processors, and all messages are received in the round in which they are sent.

**3. Exponential Information Gathering Protocols.** We start by describing the exponential protocol due to Bar-Noy *et al.* [2] (*Exponential Information Gathering*—EIG), which is closely related to the original protocol of Pease, Shostak and Lamport [22]. Our final protocol will be obtained by a sequence of transformations to this protocol, based on distinct observations. Our description of the EIG protocol below is essentially taken from Bar Noy *et al.* [2].

In the first round of the EIG protocol for Byzantine agreement, each processor broadcasts its initial value $v_i$ to all other processors. In each of the following $t$ rounds, every processor broadcasts all of the information it received in the latest round. At the end of $t+1$ rounds each processor computes a decision value based on the information it has gathered, decides on this value and halts.

We now describe the protocol in greater detail. Each processor incrementally constructs a tree-based data structure which we will call an EIG *tree*. We consider the root of the tree to be a node of *depth* 0, and inductively define the depth of a node to be greater by one than the depth of its parent. We will only be interested in EIG trees of depth at most $t+1$. In the EIG tree, a node of depth $r$ has $n-r$ children. Thus, in particular, the root has $n$ children. The edges of the EIG tree are labelled with processor names as follows. The outgoing edges of the root are labelled $1, \ldots, n$, respectively. A node of depth $r \geq 1$ has an edge labelled $i$ for every processor name $i$ that does not appear on the path leading from the root to the node. Notice that with this definition no label appears twice on a path from the root to a leaf. It follows that the sequence of labels on the path from the root to a given node uniquely determines this node. Moreover, there is a 1-1 correspondence between strings $\sigma$ of up to $t+1$ distinct processor names and the nodes in an EIG tree of depth $t+1$. We will thus regard such a string as the *name* of the corresponding node, and refer to nodes by such names. The root is named by $\lambda$, which stands for the empty string. Notice that the length of the string $\sigma$, which we shall denote by $|\sigma|$, coincides with the depth of $\sigma$.

We shall ultimately associate two values with each node $\sigma$ in a processor $i$'s tree: a *stored value*, denoted by $\mathtt{tree}(\sigma)$, and a *resolved value* denoted by $\mathtt{res}(\sigma)$. When we need to specify the particular processor $i$ in whose tree these values appear, we denote them by $\mathtt{tree}_i(\sigma)$ and $\mathtt{res}_i(\sigma)$ respectively. The stored values are assigned during the $t+1$ rounds of information exchange between the processors, while the resolved values are computed at the end, in order to determine the decision value. The manner in which these values are arrived at is described below. A node $\sigma j$ is said to *correspond* to the processor $j$ whose name labels the edge leading to the node from its parent $\sigma$. We call a node of the EIG tree *correct* if it corresponds to a nonfaulty processor.

Each processor $i$ initially stores its initial value $v_i$ in $\mathtt{tree}_i(\lambda)$—at the root of its EIG tree. In the first round of the EIG protocol a processor will send the value stored in its root to all $n$ processors (including itself, although of course this message need not actually be sent). For every processor $j$, the value that $j$ sends to $i$ in the first round will be stored in the node $\langle j \rangle$ of $\mathtt{tree}_i$ (a default value of 0 will be stored in $\mathtt{tree}_i(\langle j \rangle)$ in case processor $j$ does not receive a legitimate message with a value from $j$). In each subsequent round every processor sends to all other processors the level of its tree most recently filled in. (Note that a faulty processor may of course send different values than the ones it should send, and also may send conflicting messages

to different processors in the same round.) The messages received are broken up and used to form a new level in the processor's tree as follows: If $j$'s message reports the value $v$ for the node $\sigma$, and $j$ does not appear in $\sigma$, then the value $v$ will be stored at the node $\sigma j$. Again, we store a default value of 0 in $\sigma j$ in case $j$ did not report a legitimate value for $\sigma$ (that is, if $j$ did not send a message that is in the syntax appropriate for messages sent in this round). Intuitively, the node $\sigma j$ in $\mathtt{tree}_i$ stores the value that $j$ claims in its message to $i$ to have stored in $\mathtt{tree}_j(\sigma)$. Notice that the single message received by $i$ from $j$ in a given round reports on the values of many nodes in $j$'s tree, and will be used to update many different nodes in $\mathtt{tree}_i$.

We consider a node to be *created* in the round in which a value is stored in the node. In the first stage of the EIG protocol, information is gathered for $t+1$ rounds, until all nodes of depth up to $t+1$ are created. At that point each processor computes a value for the tree, by applying the recursive computation of $\mathtt{res}_i(\cdot)$ to the root $\lambda$. The processor then "decides" on the value of $\mathtt{res}(\lambda)$, and halts. For the purpose of the current section and the next one, the particular function used for $\mathtt{res}(\cdot)$ is *recursive majority voting*, defined as follows:

$$\mathtt{res}_i(\sigma) = \begin{cases} \mathtt{tree}_i(\sigma) & \text{if } \sigma \text{ is a leaf;} \\ 1 & \text{if majority of } \mathtt{res}_i(\sigma j) \text{ are 1;} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\mathtt{res}_i(\sigma)$ computes the value of the recursive majority of the descendants of the node $\sigma$ in $\mathtt{tree}_i$. This completes the description of the EIG protocol. Bar Noy *et al.* prove the following:

THEOREM 3.1 (BAR-NOY *et al.* [2]). *Protocol* EIG *solves the Byzantine agreement problem for* $n > 3t$.

**4. Polynomial Protocols for $n > 4t$.** While the EIG protocol is a correct $(t+1)$-round BA protocol for $n > 3t$, it has a major drawback in requiring messages and local memory of exponential size. In fact, *every* run of this protocol is exponential. Nevertheless, the clean structure provided by the EIG tree will allow us to apply a sequence of transformations to this protocol, and to finally arrive at a polynomial protocol. In this section we shall present the basic observations that lead to a polynomial protocol in the case of $n > 4t$. In Section 5 we shall highlight the problems and discuss the modifications required in the case of $n > 3t$. This will require a careful analysis and new techniques. In particular, we shall introduce a new class of resolve functions, which are instrumental in achieving a polynomial protocol for $n > 3t$.

**4.1. Predicting resolved values.** The first step is based on an observation due to Moses and Waarts in [21]: A processor $i$ can often determine early on what the value of $\mathtt{res}_i(\sigma)$ in the EIG protocol will be. This is commonly called *prediction.* Roughly speaking, once $i$ is able to predict the resolved value of $\sigma$, it can stop gathering information about the whole subtree of $\mathtt{tree}_i$ rooted at $\sigma$. One basis for prediction is the following lemma that is used in the proof of Theorem 3.1:

LEMMA 4.1 (BDDS). *Let $n > 3t$, let $i$ and $j$ be nonfaulty processors, and let $\sigma$ be a node of depth $|\sigma| \leq t$ such that $j$ does not appear in $\sigma$. Then at the end of round $t+1$ we have $\mathtt{res}_i(\sigma j) = \mathtt{tree}_i(\sigma j) = \mathtt{tree}_j(\sigma)$.*

As discussed in [21] if, for example, *majority* $+t-1$ children of $\sigma$ store the same value $v$ in $\mathtt{tree}_i$, then $i$ knows that at least majority of children of $\sigma$ are correct and store $v$. Hence, by Lemma 4.1 we get that the majority of $\mathtt{res}_i(\sigma j)$ values will

be $v$. The definition of the resolve function now implies that we will end up with $\mathtt{res}_i(\sigma) = v$. Following [21], we will say in this case that the value of $\sigma$ is *fixed to* $v$ *for* $i$ once $\sigma$'s children in $\mathtt{tree}_i$ have been created. Another case in which the resolved value of a node can be predicted is when a majority of its children become fixed to the same value. We also define a node $\sigma$ to be *closed* in $\mathtt{tree}_i$ at the end of round $r$ if either $\sigma$ or one of its ancestors is fixed in $\mathtt{tree}_i$ at that time. The properties of prediction in the case of $n > 4t$ are described in the following two statements.

LEMMA 4.2 (MW). *Assume $n > 4t$, let $\sigma$ be a correct node with $|\sigma| \leq t$, and let $i$ be a nonfaulty processor. Then $\sigma$ is closed in $\mathtt{tree}_i$ by the end of round $|\sigma| + 1$.*

COROLLARY 4.3 (MW). *Assume $n > 4t$ and let $i$ and $j$ be nonfaulty processors. In the* EIG *protocol, if $\sigma$ is fixed for $i$ at the end of round $r$, then $\sigma$ is closed in $\mathtt{tree}_j$ at the end of round $r + 1$ at the latest.*

An immediate implication of Corollary 4.3 is that a nonfaulty processor need report on descendants of a node for at most one round after it can determine that the node is fixed to some value. No further information it can send about the subtree of that node will help anyone determine its final resolved value. In particular, once the root becomes fixed in $\mathtt{tree}_i$, processor $i$ knows the decision value and needs to send information for at most one more round. [21] uses this to devise an early-stopping BA protocol for $n > 4t$. This protocol is still exponential, although it is significantly more efficient than the one that constructs the full EIG tree. Moreover, there are many cases in which the tree is polynomial in size. In fact, in the common case in which no failures arise, the protocol ends after two rounds, and the amount of communication sent by a processor is $O(n)$ in the case of single-source BA and $O(n^2)$ in the consensus case.

Lemma 4.2 provides some insight into the relationship between the size of the EIG trees (and hence the complexity of computation and communication) in a given run of the early-stopping BA algorithm and the behavior of the faulty processors during that run. Let us call a node $\sigma$ *corrupted in* $\mathtt{tree}_i$ if $\sigma$ is not closed in $\mathtt{tree}_i$ by the end of round $|\sigma|+1$. Clearly, a non-corrupted node can have at most $n$ children in the tree. In addition, it is not hard to check that a corrupted node can account for the need to store its grandchildren, but not for any later descendants. The total size of an EIG tree thus becomes roughly $O(n^2C)$, where $C$ denotes the number of corrupted nodes in the tree. It follows that if we are able to reduce the number of corrupted nodes, then the trees will shrink, and hence also the communication. (Indeed, the protocol of [21] managed, for $n > 8t$, to ensure that no processor is able to corrupt more than one node overall, and thus in all trees $C \leq t$.) A central tool in reducing the number of corrupted nodes turns out to be the detection and masking of faulty processors, which we now turn to discuss.

**4.2. Detecting and masking failures.** Lemma 4.2 implies that if $\sigma j$ is corrupted in $\mathtt{tree}_i$, then processor $i$ can detect that $j$ is faulty immediately after it receives values for $\sigma j$'s children. Indeed, we shall see later on some other instances in which a processor $i$ can detect another processor as faulty. Detecting failures is useful because of the following observation, which was first made by Dolev Reischuk and Strong [13], and later used in various ways in [8, 2, 21, 3] and others: Since faulty processors can send arbitrary messages, once we detect a processor as being faulty, we can act as if it sends us particular messages that may be to our advantage, ignoring what it actually sends us. This is called *fault masking*. One particularly simple form of fault masking is to regard a processor detected as faulty to send us a fixed value (e.g., 0) for all nodes of interest. As we shall see, this can help reduce the number

of corrupted nodes. Once all nonfaulty processors mask a given processor $z$, we say that $z$ is *disabled*. Notice, in particular, that once $z$ is disabled, nodes of the form $\sigma z$ can no longer be corrupted. They become fixed to the value being used for masking at the end of round $|\sigma z| + 1$.

Corollary 4.3 implies that unless $\sigma j$ is corrupted in all of the correct processors' trees, it will become closed (in all trees) at the end of round $|\sigma j| + 2$. It follows that in order for the subtree of $\sigma j$ to grow to a substantial size in some processor's tree, the node $\sigma j$ must be corrupted in *all* nonfaulty processors' trees. We will call a node *universally corrupted* if it is corrupted in all nonfaulty processors' trees. Another consequence of Corollary 4.3 is that the final size of an EIG tree is polynomial if and only if the number of universally corrupted nodes in the tree is polynomial. This follows from the fact that if the subtree of a universally corrupted node has more than $n^3$ nodes, at least one of the node's children must also be universally corrupted. If a node $\sigma j$ is universally corrupted, then everyone detects $j$ as faulty at the end of round $|\sigma j| + 1$, and $j$ becomes disabled from that point on. It cannot corrupt nodes in later rounds. It follows that when $n > 4t$, fault masking allows a faulty processor to universally corrupt nodes only during a single round. Moreover, Lemma 4.2 implies that if neither the node $\sigma j$ nor any of its ancestors is fixed in $\texttt{tree}_i$ by the end of round $|\sigma j|$, then all processor names appearing in the string $\sigma$ denote faulty processors. This implies that once we employ early stopping and fault masking in the case of $n > 4t$, at least $3t$ of the children of any node of interest are nonfaulty. Moreover, at most $t - |\sigma|$ children of a node $\sigma j$ can be faulty. As a result, if processor $i$ finds more than $t - |\sigma|$ values among the children of $\sigma j$ that differ from $\texttt{tree}_i(\sigma j)$, it can detect that $j$ is faulty.

If no faulty processor universally corrupts a node in a given round $r$, then all nonfaulty processors decide by the end of round $r + 1$. If only one processor universally corrupts nodes in each round (and we employ early stopping and fault masking), then the total size of the tree is polynomial. Following [3], we shall call ESFM (for "Early Stopping with Fault Masking") the protocol resulting from the combined application of the prediction and fault masking techniques to EIG. The ESFM protocol significantly reduces the size of the EIG tree processors construct.[1] However, as discussed in [21], the trees may still grow exponential in general because (i) more than one processor can corrupt nodes in a given round, and (ii) a processor can corrupt many nodes in a given round. The final observation that will yield a polynomial protocol, at least in the case of $n > 4t$, is that when many processors universally corrupt nodes in a given round, it is possible for the nonfaulty processors to detect that something "fishy" is going on, and to decide to stop. How to do so is the subject of our next subsection.

**4.3. Monitor voting.** An important new idea was introduced by Berman and Garay in [3]. They proposed to have each processor observe the size of its tree, and when the size exceeds a certain threshold, to "vote" to stop the whole agreement procedure. Thus, roughly speaking, they started a new instance of Byzantine agreement in every round, in which a processor's initial value reflects whether or not the processor is in favor of stopping the process (and deciding on a default value). In order for any processor's tree to become exponential, all processors' trees must grow beyond any polynomial threshold. Should this happen, all nonfaulty processors would

[1] Using both techniques—prediction and fault masking—simultaneously reduces the tree from size $O(n^t)$ to $O(c^t)$[21]. In [4] a more sophisticated masking technique yields a similar result for $n > 3t$.

vote in favor of stopping and deciding on the default, and this instance of the agreement protocol would stop in two rounds. In the [3] protocol, once such an agreement is attained, every nonfaulty processor decides on the default and halts, terminating its participation in all Byzantine agreement instances underway.[2] We think of the agreement instances initiated in every round of the [3] protocol as processes that *monitor* the run, ensuring that trees do not grow too much. Indeed, Berman and Garay showed that when performed in a careful way, the monitor voting technique yields the following:

THEOREM 4.4 (BERMAN AND GARAY [3]). *By applying monitor voting to the* ESFM *tree, it is possible to obtain a* $(t+1)$*-round polynomial-time Byzantine agreement algorithm for* $n > 4t$.

...

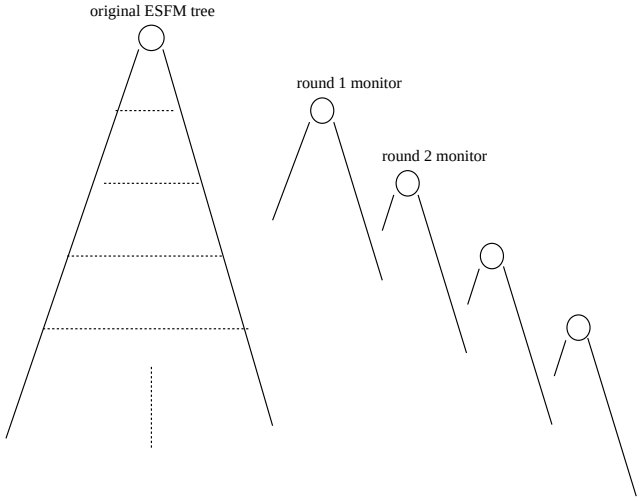


FIG. 1. *The monitor voting technique.*

The technique is illustrated in Figure 1. Monitor voting introduces a number of new subtleties:

(i) Once there are a number of agreement processes underway, we must be careful to ensure that processors use the individual agreements' values in a consistent way when determining their ultimate decision value. If, for example, one processor may decide 0 because its original information gathering tree stopped with value 0, while the other decides 1 because one of the monitors stopped with value 1, we are in trouble.

(ii) As monitor processes are initiated in different rounds of the execution, we need to take action to ensure that they all halt by time $t + 1$. Otherwise, we may run into consistency problems as described above, when we are forced to decide at

[2] This technique was called *Cloture Votes* by the authors, since it resembles a procedure in the U.S. Senate in which a vote can be called on whether the discussion on a given topic has been going on for too long.

time $t + 1$.

(iii) It may seem desirable to exclude faulty processors from participating in a monitor, in order to ensure the monitor halts in time. However, deciding that a given processor is faulty is as hard as Byzantine agreement itself. Techniques of [12] can be used to show that we can not exclude processors "on the fly," as a result of failures being detected.

The main tool that can be used to limit the damage caused by faulty processors is to use information about failures that a processor has gained in one agreement process to mask faulty processors in all agreement processes. It is crucial, however, to ensure that sufficiently many processors are *disabled* before a processor can vote in favor of stopping in a monitor agreement process. In fact, we shall use the following rule for determining the initial value of processor $i$ in monitor process $\mathtt{M}^r$, which is the monitor agreement process initiated in round $r$:

**Monitor-vote:** Processor $i$ will vote 1 on $\mathtt{M}^r$ if it has detected that $r - 1$ faulty processors have corrupted nodes and are disabled by the end of round $r - 1$; it will vote 0 on $\mathtt{M}^r$ otherwise.

In the case of $n > 4t$, it is possible to show that for the information gathering trees to grow large, the number of disabled processors must grow sufficiently fast that it would enable every nonfaulty processor to eventually vote 1 using this rule. This is the basis of the solution for $n > 4t$ given in [3]. Extending this idea to $n > 3t$, however, seems to be problematic, because the faulty processors have greater powers to corrupt nodes when $4t \geq n > 3t$.

**5. Deriving a Polynomial Protocol for $n > 3t$.** Our goal in this paper is to describe a polynomial BA protocol for $n > 3t$ that is guaranteed to terminate in $t+1$ rounds. Intuitively, the protocol follows the monitor voting approach presented in the previous section. In the case of $n > 3t$, however, most of the properties used when $n > 4t$ no longer hold. First of all, using the resolve function defined in Section 4, the analogues of Lemma 4.2 and Corollary 4.3 fail. I.e., it is possible to show that when $n < 4t$ the majority function does not guarantee quick prediction of the value $\mathtt{res}(\sigma)$ even for correct nodes $\sigma$. (Indeed, we know of no constant number of generations of descendants for which `res` guarantees prediction of correct nodes.) In [25] Waarts presented prediction rules for a resolve function introduced by Bar-Noy *et al.* [2], and showed that they guarantee prediction in two rounds. In the case of $n > 4t$ we had the property that exponential size of trees requires many faulty processors to expose themselves (and hence become disabled) early in the run. Intuitively, we say that some faulty processors must be "wasted" for the adversary to be able to cause trees to grow exponential. This was crucial for the monitor voting scheme to succeed in keeping the trees polynomial. As we shall see, the fact that it sometimes takes nodes two rounds to fix when $n > 3t$ implies that there are often cases in which the tree can grow without requiring such waste to occur. The function used by Waarts in [25] is ternary, and it occasionally provides an "undecided" value for an internal node. This introduces enough "slack" into the information conveyed by the resolve function, that we were unable to use it successfully in the case of $n > 3t$. It seems possible for the adversary to cause the trees to grow exponential when that resolve function is used, without the number of disabled processors ever becoming large enough to be "caught" via monitor voting. A major focus of this paper is on deriving a resolve function and corresponding prediction and fault detection rules that will guarantee that exponential growth of the agreement trees will require sufficient waste to enable monitor voting to detect the problem and halt.

In the rest of this section we introduce a radically different class of resolve functions. This class of functions is then used in Section 6 as the basis of a protocol for a slightly generalized version of Byzantine agreement. We end up chosing the precise function from this class in such a way that it guarantees that for the size of the EIG trees to grow beyond a polynomial bound, faulty processors must be disabled at a rate that is greater than one per round. This is the key point that enables us to perform monitor voting for the case of $n > 3t$ in the spirit of the monitor voting we performed for $n > 4t$.

**5.1. A general class of resolve functions.** Let us denote the set $\{0, \ldots, j\}$ by $[j]$. For every function $F : [t] \to [n]$, we define a resolve function $\mathtt{res}^F$ as follows:

$$\mathtt{res}^F(\sigma) = \begin{cases} \mathtt{tree}(\sigma) & \text{if } \sigma \text{ is a leaf;} \\ 1 & \text{if at least } F(|\sigma|) \text{ of } \mathtt{res}^F(\sigma j) \text{ are 1;} \\ 0 & \text{otherwise.} \end{cases}$$

Notice that $\mathtt{res}^F(\sigma) = 0$ if and only if $\#\{j : \mathtt{res}^F(\sigma j) = 0\} \geq n - |\sigma| + 1 - F(|\sigma|)$. Moreover, notice that the original $\mathtt{res}$ function of [2] is simply $\mathtt{res}^M$, where $M$ is the majority function. Let us now turn to a straightforward observation regarding functions of this type. Let a *cut* through the EIG tree be a set $C$ of nodes that intersects every path from the root to a leaf exactly once. One important property of resolve functions is that the values of a resolve function on the nodes of a cut uniquely determine its values on all nodes above it:

LEMMA 5.1. *Let $C$ be a cut of the* EIG *tree, and let $F : [t] \to [n]$. Then the values of $\mathtt{res}^F$ on the nodes of $C$ uniquely determine the values of $\mathtt{res}^F(\sigma)$, for all nodes $\sigma$ that are ancestors of nodes in $C$.*

*Proof.* Notice that because $C$ is a cut, if a node $\sigma$ is an ancestor of a node in $C$, then every one of $\sigma$'s children is either in $C$ or an ancestor of a node in $C$. The proof is by induction on the height of $\sigma$ above $C$. Assume that $\sigma$ is an ancestor of a node in $C$ and that the values of $\mathtt{res}^F$ on the nodes of $C$ uniquely determine the values of all of $\sigma$'s children $\sigma j$. Since $\mathtt{res}^F(\sigma)$ depends only on the values of $\mathtt{res}^F$ on $\sigma$'s children, and their values are uniquely determined by the values of $\mathtt{res}^F$ on $C$, we are done. □

This lemma has an immediate corollary that we shall find most useful in the sequel.

COROLLARY 5.2. *Let $F : [t] \to [n]$. If, for all nodes $\sigma$ of some cut $C$ it is the case that $\mathtt{res}^F(\sigma)$ has the same value in all correct processors' trees, then the values of $\mathtt{res}^F$ on every node above $C$ are the same in all trees. In particular, this applies to $\mathtt{res}^F(\lambda)$, the resolved value for the root.*

We are now in a position to identify a large class of functions $F$ that can serve as resolve functions for Byzantine agreement protocols. We say that $F$ is a *sound* resolve function for Byzantine agreement if applying $\mathtt{res}^F$ to the complete EIG trees of nonfaulty processors is guaranteed to fulfill the requirements of Byzantine agreement. (Notice that Decision is trivial in this case, since it is assumed that a processor decides on $\mathtt{res}^F(\lambda)$; thus, the claim actually concerns the Agreement and Validity properties).

LEMMA 5.3. *If $F : [t] \to [n]$ satisfies $t + 1 \leq F(r) \leq n - t - r$, then $\mathtt{res}^F$ is a sound resolve function for Byzantine agreement for $n > 3t$.*

*Proof.* The proof proceeds along the lines of the proof of $\mathtt{res}^M$ (the Majority resolve function) from [BDDS]. First notice that we are only interested in applying this rule for depths $r$ satisfying $r \leq t$. In this case, if $n > 3t$ then indeed $n - t - r \geq t + 1$, so such functions $F(r)$ exist.

Let $h_L(\sigma) = t + 1 - |\sigma|$ be the height of the node $\sigma$ above the leaves. First, we show by induction on $h_L(\sigma)$ that every correct node $\sigma$ is resolved to $\mathtt{tree}(\sigma)$. For a leaf $\sigma$ ($h_L = 0$) this is immediate from the definition of $\mathtt{res}^F$. Assume the claim is true for all children of $\sigma$, and that $\sigma$ is a correct node with $h_L(\sigma) \geq 1$. It follows that at least $n-t-|\sigma|$ of $\sigma$'s children $j$ are correct. Moreover, since $\sigma$ is correct, all correct processors $j \notin \sigma$ receive and echo an identical value for $\sigma$, so that $\mathtt{tree}(\sigma j) = \mathtt{tree}(\sigma)$. By the inductive hypothesis, we thus have $\#\{j : \mathtt{res}^F(\sigma j) = \mathtt{tree}(\sigma)\} \geq n - t - |\sigma|$. The claim is obtained by observing that $n - t - |\sigma| \geq F(|\sigma|)$, which covers the case $\mathtt{tree}(\sigma) = 1$, and $n - t - |\sigma| = n - |\sigma| + 1 - (t+1) \geq n - |\sigma| + 1 - F(|\sigma|)$, which covers the case of $\mathtt{tree}(\sigma) = 0$. This completes the inductive argument.

To complete the proof, notice that a tree of depth $t+1$ has at least one correct node on every path from the root to the leaves. It follows that there is a cut $C$ through the tree consisting of correct nodes only. For correct nodes $\sigma$ (and in particular all nodes $\sigma \in C$) we have just shown that $\mathtt{res}^F(\sigma)$ is the same in all correct processors' trees. It follows by Corollary 5.2 that $\mathtt{res}^F(\lambda)$ is the same in all trees, and we are done. □

Notice that Lemma 5.3 allows a wide range of functions $F(r)$, including the natural function $M$ (majority). Our quest to lower the complexity of the algorithm requires that we do not expand the whole EIG tree. Rather, we wish to use information we gather incrementally in order to be able to predict the resolved values of nodes, and thereby hopefully be able to avoid expanding their subtrees. Once every processor is able to predict the resolved values of nodes on a cut of the tree, we shall be done. Indeed, as shown in [21], when $n > 4t$ the function $\mathtt{res}^M$ guarantees that the resolved values of correct nodes can always be predicted once values of their children are stored. Unfortunately, when $n < 4t$ majority does not guarantee quick prediction of the value of $\mathtt{res}^M(\sigma)$ even for correct nodes $\sigma$. We shall thus seek other functions that will allow effective prediction.

We shall be most interested in a particular type of resolve function satisfying the conditions of Lemma 5.3 which, roughly speaking, will alternate between favoring the value 0 and favoring the value 1 in consecutive rounds. (Intuitively, a favored value is one that requires a minimal amount of support in a given round in order to force a parent to be resolved to this value, while the opposite value requires a substantial amount of support.) A major consequence of this "flipping" function will be that the resolved values of correct nodes can be predicted in two rounds at the latest. More specifically, the alternating behavior will be controlled by the *parity* of the round. We let $\mathtt{par}(r)$ denote the parity of $r$. Thus, $\mathtt{par}(0) = 0$, $\mathtt{par}(1) = 1$, $\mathtt{par}(2) = 0$, etc.

Another important property of our function is that it will only be defined for an appropriate portion of the EIG tree, and not for the complete tree. More specifically, let us define a node $\sigma$ to be *righteous* if $\sigma$ is a correct node, and all of its ancestors are faulty. (In other words, a righteous node is the first correct node on some path from the root to a leaf.) Clearly, the identity of the faulty processors in a given run completely determines which nodes of the tree are righteous. An important property of the EIG tree is that given any choice of up to $t$ faulty processors, the set of righteous nodes forms a cut in the EIG tree. Given a set $B$ of up to $t$ faulty processors, we define the *righteous subtree* of the EIG tree (with respect to $B$) to be the tree whose leaves are the righteous nodes, and whose internal nodes consist of the ancestors of righteous nodes. Clearly, different choices for $B$ yield different righteous subtrees. (In the sequel, the set $B$ will be implicit, and will consist of the faulty processors in the run under discussion.) Our goal will be to devise a mechanism by

which, roughly speaking, righteous nodes will be resolved to their stored values in all nonfaulty processors' trees. Since the righteous nodes are in particular correct nodes, this means that they will be resolved to the same value in all trees. Once righteous nodes resolve to the same values, we can use them in the role of leaves in the definition of a new resolve function. We thus define a resolve function $\texttt{right}^F$ on nodes $\sigma$ of the righteous subtree as follows:

$$\texttt{right}_i^F(\sigma) = \begin{cases} \texttt{tree}_i(\sigma) & \text{if } \sigma \text{ is righteous;} \\ \texttt{par}(|\sigma|) & \text{if } \texttt{right}_i^F(\sigma j) = \texttt{par}(|\sigma|) \text{ for at least} \\ & F(|\sigma|) \text{ nodes } \sigma j\text{; and} \\ 1 - \texttt{par}(|\sigma|) & \text{otherwise.} \end{cases}$$

Notice that $\texttt{right}_i^F(\sigma)$ is undefined for nodes $\sigma$ outside the righteous subtree. Since, for a righteous node $\sigma$, we are guaranteed that $\texttt{tree}_i(\sigma) = \texttt{tree}_j(\sigma)$ holds for all (nonfaulty) $i$ and $j$, it is the case that $\texttt{right}_i^F = \texttt{right}_j^F$. Since the righteous nodes form a cut in the tree, it follows by Lemma 5.1 that $\texttt{right}_i^F$ is independent of $i$. Therefore, we shall henceforth drop the subscript and refer to this function by $\texttt{right}^F$. We now have:

PROPOSITION 5.4. *$\texttt{right}^F$ satisfies the Agreement and Validity conditions of Byzantine agreement for every resolve function $F : [t] \to [n]$.*

The definition of $\texttt{right}^F$ seems to have a drawback: Processors are in general unable to tell a righteous node from a non-righteous one. So how can $\texttt{right}^F$ be used by the processors? Intuitively, our plan is to present "fixing" rules that would predict the $\texttt{right}^F$ value of a node, provided the node is in the righteous subtree. For nodes outside the righteous subtree the $\texttt{right}^F$ value is undefined, and we give no guarantee on what values our fixing rules may give. This will not cause problems, because our rules will have the property that for every righteous node $\sigma$, either $\sigma$ or one of its ancestors will be fixed by the time values for $\sigma$'s grandchildren are stored in the tree (i.e., by the end of round $\min(t+1, |\sigma|+2)$. Thus, our scheme will guarantee that all nodes of the righteous subtree, including the root, become fixed in all trees by the end of round $t+1$, if not earlier. Moreover, the nodes of the righteous subtree, when fixed to a value, will be fixed to their $\texttt{right}^F$ value. We shall now turn to formalize this intuition.

**6. $\Delta$-agreement.** The monitor processes we talk about are agreement protocols that closely resemble ordinary Byzantine agreement, except for the following differences. (a) A monitor process is initiated in a state in which each nonfaulty processor $i$ has a set $\mathcal{F}_i$ of processors that $i$ has already detected as faulty, and is masking throughout the monitor. (b) The set of faulty processors used by processor $i$ in each of the monitors is obtained based on fault detection performed by $i$ in *all* active agreement processes. (c) Finally, we will associate with a monitor `M` a parameter $\Delta < t$ which, roughly speaking, is a lower bound on the number of initially disabled faulty processors. This gives rise to a slight generalization of Byzantine agreement, that we shall call $\Delta$-agreement.

We define $\Delta$-agreement more formally as follows. As in Byzantine agreement, in an instance of $\Delta$-agreement, each nonfaulty processor $i$ starts out with an initial value $v_i \in \{0,1\}$. In addition, every nonfaulty processor $i$ starts out with a set $\mathcal{F}_i$ of faulty processes. If all nonfaulty processors start out with an initial vote of 0, the parameter $\Delta$ plays no role. If, however, at least one nonfaulty processor votes 1, then at least $\Delta$ faulty processors are initially disabled. Let $\mathcal{D} \stackrel{\text{def}}{=} \bigcap_i \mathcal{F}_i$ denote the set

of initially disabled processors. In $\Delta$-agreement, we are guaranteed that if at least one nonfaulty processor votes 1, then $\#\mathcal{D} \geq \Delta$. (Thus, if all nonfaulty processors vote 0, no guarantee about the size of $\mathcal{D}$ is given.) Strictly speaking, an instance of $\Delta$-agreement has two additional parameters, $n$ and $t$, where as usual $n$ is the total number of processors, and $t$ is an upper bound on the total number of faulty processors (including the ones in $\mathcal{D}$ and in the $\mathcal{F}_i$'s). Rather than working explicitly with $(n, t, \Delta)$-agreement, we shall continue to talk about $\Delta$-agreement, keeping $n$ and $t$ implicit and assuming that $n$ and $t$ satisfy $n \geq 3t + 1$, while $\Delta < t$. The Decision, Agreement and Validity requirements are as in the case of Byzantine agreement defined in Section 2. Notice that the standard variant of Byzantine agreement that we have been considering can be viewed as an instance of $\Delta$-agreement, where $\Delta = 0$ and $\mathcal{F}_i = \emptyset$ for every nonfaulty processor $i$.

**6.1. Basic structure of the $\Delta$-EIG protocol.** Our purpose in this section will be to describe a protocol for $\Delta$-agreement, which we shall call the $\Delta$-EIG protocol. In later sections we shall discuss how to combine a number of these protocols via monitor voting to obtain an efficient solution to Byzantine agreement.

Our $\Delta$-EIG protocol for a single instance of $\Delta$-agreement will be based on a number of components. We now discuss some of them. First of all, the $\Delta$-EIG protocol will operate on an EIG tree of depth $t + 1 - \Delta$ (as opposed to depth $t + 1$ in the standard EIG protocol). This will be essential, as we want to be able to complete a run of this protocol within $t+1-\Delta$ rounds. In addition, we shall have every processor $i$ maintain a set of processors it has detected as faulty. Let $\mathcal{F}_i(r)$ denote the set of faulty processors detected by $i$ in the first $r$ rounds. Thus, in particular, $\mathcal{F}_i(0) = \mathcal{F}_i$, and the sets $\mathcal{F}_i(r)$ grow monotonically over time (i.e., $\mathcal{F}_i(r + 1) \supseteq \mathcal{F}_i(r)$). These sets will be used by the processors both for masking values in their own trees, and for reporting on masked nodes. Rather than processor $i$ sending in round $r + 1$ separate reports of masked values for each node $\tau z$ corresponding to processors $z \in \mathcal{F}_i(r)$, we shall have $i$ send a report of the form $\texttt{mask}(i, z)$ in the first round following the one in which it detects $z$ to be faulty. All processors that receive this report will, from then on, act as if $i$ actually sends separate masked reports for such nodes. In fact, we shall abuse the language slightly, and consider a processor $i$ that issues a $\texttt{mask}(i, z)$ report in round $r$ as if it "reports" masked values for all nodes $\sigma z$ with $|\sigma z| \geq r$. A processor will keep track of $\texttt{mask}(i, z)$ reports it receives, and will store masked values in nodes accordingly on $i$'s behalf, as specified below. In addition to saving in communication, this will allow a processor to detect failures based on reports its receives, and will, later on, make it easier for $i$ to estimate the number of disabled processors.

Formally, processors will send messages according to the following rule:

- **Sending:** In a given round $r + 1$, a processor $i$ sends all other processors a message consisting of two components. In the first component, the message contains reports $\texttt{mask}(i, z)$ on the processors $z \in \mathcal{F}_i(r) \setminus \mathcal{F}_i(r - 1)$ that $i$ has just discovered as faulty. For completeness, we formally define $\mathcal{F}_i(-1) = \emptyset$, so that in the first round $i$ reports that it is masking the processors in $\mathcal{F}_i(0) = \mathcal{F}_i$. The second component of the message consists of pairs $\langle \sigma j; v \rangle$, where $v = \texttt{tree}_i(\sigma j)$, for all nodes $\sigma j$ of depth $r$ such that $j \notin \mathcal{F}_i(r)$.

Upon receiving the messages sent to it in round $r$, processor $i$ will record and mask values as follows:

- **Recording and masking:**

1. Processor $i$ appends every $\texttt{mask}(z, j)$ report it receives in round $r$ to a list of mask reports that it maintains;

2. Processor $i$ records values in $\texttt{tree}_i$ according to

1. If $j \notin \mathcal{F}_i(r-1)$, then $\texttt{tree}_i(\tau j)$, for a node $\tau j$ of depth $r$, is the value reported by $j$ for $\tau$ in round $r$. In particular, if $\tau = \sigma z$ for some $z$ such that $i$ has received a $\texttt{mask}(j, z)$ report from $j$ in one of the first $r$ rounds, then $\texttt{tree}_i(\tau j) = \texttt{par}(|\tau j|)$; and

2. If $z \in \mathcal{F}_i(r-1)$ and $\tau z$ is a node of depth $r$, then $\texttt{tree}_i(\tau z) = \texttt{par}(|\tau|)$.

In particular, this means that values of nodes corresponding to initially detected failures are always masked.

The set $\mathcal{F}_i$ is updated in the following manner:

- **Fault detection:** $\mathcal{F}_i(r)$ is obtained by adding to $\mathcal{F}_i(r-1)$ any new processor failure discovered by applying the fault detection rules FD0–FD3 described in Section 6.3 below to $\texttt{tree}_i$ after the Recording and masking steps have taken place. Since, as discussed later on, the fault detection rules will be computable in a fairly efficient manner, this whole step is feasible. We remark that it would be possible to use the new failures detected in the last step in order to mask additional nodes, and then perhaps perform the fault detection step again. Indeed, this process could be repeated until no new failures would be discovered. For the sake of simplicity, we choose not to do so. The failures discovered in round $r$ will affect processors' messages and processing from round $r+1$ on.

Faulty processors will be assumed to be discovered according to a set of sound fault discovery rules. The only thing we require of this set of rules is that it should include the rules FD0–FD3 described in Section 6.3 below. The soundness of the rules implies that one invariant of our algorithm will be:

- **Soundness:** If $i$ and $j$ are nonfaulty, then $j \notin \mathcal{F}_i(r)$ holds for all rounds $r$.

Given the Soundness invariant, the Sending and the Recording and masking rules guarantee that $\texttt{tree}_i(\sigma) = \texttt{tree}_j(\sigma)$ will hold for every correct node $\sigma$ of depth at most $t+1-\Delta$, and nonfaulty processors $i$ and $j$. It follows that the values of the function $\texttt{right}^F$ will continue to be independent of the tree in which they are computed.

Prediction will be handled by a set of fixing rules Fx1–Fx3 described in Section 6.2 below. These rules will determine when a node $\sigma$ is said to be *fixed to value $v$ in* $\texttt{tree}_i$. Recall that we defined a node $\sigma$ to be *closed in* $\texttt{tree}_i$ at the end of round $r$ if either $\sigma$ or one of its ancestors is fixed in $\texttt{tree}_i$ at that point.

Finally, we shall use a simple rule for deciding on a value and for halting in the basic $\Delta$-EIG protocol:

- **Deciding:** When the root $\lambda$ becomes fixed to a value $v$ in $\texttt{tree}_i$, processor $i$ decides on value $v$.

- **Halting:** Processor $i$ continues to record information, perform fault detection, and report on values until the end of round $t+1-\Delta$, at which point it halts.

We shall show in Lemmas 6.7 and Corollary 6.9 that the root cannot be fixed both to 0 and to 1, and that the root is guaranteed to be fixed by the end of round $t+1-\Delta$. As a result, the decision rule given above is well defined and will guarantee that $i$ will decide on a value.

To complete the description of $\Delta$-EIG, we need to describe the rules by which nodes are fixed to values, and the manner in which processors perform fault discovery. This will be the subject of the next two subsections.

**6.2. Fixing nodes to values.** We now define when a node $\sigma$ is *fixed* in $\texttt{tree}_i$ to a value $v$. This will be a major component in our protocol, and the properties of fixing, which we discuss below, will be instrumental in the development of the

algorithm. We start with a fairly abstract definition of the fixing rules, relative to a function $F : [t] \to [n]$. We shall call such a function $F$ *admissible* if it satisfies that

(i) $F(0) = F(1) = t + 1$, and

(ii) $t + r - 1 \geq F(r) \geq t - r + 2$ for $r \geq 2$.

In the sequel, we shall restrict our attention to admissible functions. A considerable amount of our analysis will be valid for admissible functions in general. Only in Section 7 will we choose a particular admissible function to be used in our final protocol. We remark that admissible functions do not necessarily conform to the conditions of Lemma 5.3. As we shall see, the fact that we shall be fixing values of nodes before the full tree is developed will allow us to go beyond the bounds of Lemma 5.3. The role of Lemma 5.3 is in motivating the development of admissible functions and our fixing rules. It will not play a role in the correctness of our protocol in the end.

The bounds used in the definition of admissible functions were chosen so that they would match the following fixing rules. Formally, a node $\sigma$ becomes *fixed in* $\texttt{tree}_i$ *to value* $v$ at the end of round $r$ if $\sigma$ was not closed at the end of round $r - 1$, and one of the following rules applies:

**Fx1:** $r = |\sigma| = t + 1 - \Delta$ and $\texttt{tree}_i(\sigma) = v$.

**Fx2:** $r = |\sigma| + 1$, $\texttt{par}(|\sigma|) = v$ and

$$\texttt{tree}_i(\sigma j) = v \text{ for at least } \begin{cases} n - t \text{ nodes } \sigma j & \text{if } \sigma = \lambda; \\ n - t - 1 \text{ nodes } \sigma j & \text{if } |\sigma| = 1; \text{ and} \\ n - t - 2 \text{ nodes } \sigma j & \text{if } |\sigma| \geq 2. \end{cases}$$

**Fx3:** Rules Fx1 and Fx2 do not apply, and either

(a) $\texttt{par}(|\sigma|) = v$ and at least $F(|\sigma|)$ of the $\sigma j$'s are fixed to $v$; or

(b) $\texttt{par}(|\sigma|) = 1 - v$ and at least $n - |\sigma| - F(|\sigma|) + 1$ (i.e., all but at most $F(|\sigma| - 1)$ of the $\sigma j$'s are fixed to $v$.

We remark that a naive bottom-up computation based on Fx1–Fx3 can be used to determine all of the fixed nodes and the values they are fixed to in a given EIG tree. Such a computation requires a number of steps at most linear in the size of the tree. The properties of the fixing rules Fx1–Fx3 will play a major role in the correctness of our ultimate protocol. We now consider some of these properties.

One immediate consequence of the definition of the above fixing rules is:

LEMMA 6.1. *Let $F$ be admissible and let $i$ be a nonfaulty processor. If all nonfaulty processors vote 0, then the root $\lambda$ is fixed to 0 in $\texttt{tree}_i$ by the end of round 1. Similarly, if all nonfaulty processors vote 1, then the root $\lambda$ is fixed to 1 in $\texttt{tree}_i$ by the end of round 2.*

*Proof.* Recall that $\texttt{par}(|\lambda|) = 0$. First assume that all nonfaulty processors vote 0. It follows that at least $n - t$ correct children of $\lambda$ in $\texttt{tree}_i$ store 0 at the end of round 1, and as a result, by rule Fx2, the root is fixed to 0 at that point. Now assume that all nonfaulty processors vote 1. The root $\lambda$ has at least $n - t$ correct children. Let $\sigma$ denote a correct child of the root $\lambda$. Thus, at the end of round 1 we have that $\texttt{tree}_i(\sigma) = 1$. The root is not fixed yet. At the end of round 2, however, $\texttt{tree}_i(\sigma j) = 1$ for at least $n - t - 1$ children of $\sigma$. Since $\texttt{par}(|\sigma|) = 1$, it follows from the second clause of Fx2 that $\sigma$ will be fixed to 1 at the end of round 2. We thus obtain that at least $n - t$ children of $\lambda$ are fixed to 1 by the end of round 2. Recall that if $F$ is admissible, then $F(0) = t + 1$. Hence, by rule Fx3 we now have that the root becomes fixed to 1 at the end of round 2, since that $F(|\lambda|) = F(0) = t + 1$, and $n - |\lambda| - F(|\lambda|) + 1 = n - (t + 1) + 1 = n - t$. □

Lemma 6.1 essentially takes care of the validity problem for $\Delta$-agreement in the case in which all nonfaulty processors have initial votes of 0. The lemma states that in this case the root in all nonfaulty processors' trees will fix to 0 by the end of round 1, and by the Decision clause of the $\Delta$-EIG protocol, all nonfaulty processors will decide 0 at that point. In the sequel, we shall therefore concern ourselves with the case in which at least one nonfaulty processor votes 1. Hence, we will be able to assume that the number of initially disabled processors is at least $\Delta$.

We now wish to use the fixing rules to show that they guarantee that nodes of the righteous subtree end up being fixed to the desirable values. In $\Delta$-agreement, however, we have processors that are initially disabled, that resemble nonfaulty processors in the fact that they cannot corrupt nodes or otherwise misbehave. As a result, finding a node corresponding to an initially disabled processor on a path from the root is quite analogous to finding a correct node. Following this observation, we shall therefore replace the definition of the righteous subtree with the analogous tree (which we shall call the *safe subtree*), and perform our analysis with respect to the new tree. Formally, we proceed as follows. Let $\mathcal{D}$ denote the set of initially disabled processors. We call a node $\sigma$ *safe* if (i) $\sigma$ is either righteous or corresponds to a processor in $\mathcal{D}$, and (ii) none of $\sigma$'s ancestors are righteous, and none of them correspond to processors in $\mathcal{D}$. Since $\#\mathcal{D} \geq \Delta$ by definition of $\Delta$-agreement, at most $t - \Delta$ processors labelling edges on a path from the root can be faulty but not from $\mathcal{D}$. It follows that every path of $t + 2 - \Delta$ nodes leading from the root must contain a safe node. In particular (since $|\lambda| = 0$), if $\sigma$ is safe, then $|\sigma| \leq t + 1 - \Delta$. Since, by definition, there can be at most one safe node on every path from the root to leaves of the EIG tree, we obtain that the safe nodes form a cut in the tree. Let us denote this cut by $C_s$. We define the *safe subtree* of an EIG tree in an execution of $\Delta$-EIG to consist of $C_s$ and all ancestors of nodes in $C_s$. We remark that $|\sigma| = t + 1 - \Delta$ can hold for a node of the safe subtree only if $\sigma$ is a safe node, and hence is either righteous or (initially) disabled. Finally, we shall use $h_s(\sigma)$, for a node $\sigma$ of the safe subtree, to denote the length of the maximal path from the node $\sigma$ to some node in $C_s$. More formally, for nodes $\sigma$ in the safe subtree, we define

$$h_s(\sigma) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \sigma \in C_s; \quad \text{and} \\ 1 + \max_j h_s(\sigma j) & \text{otherwise.} \end{cases}$$

One somewhat technical property that will serve us in the sequel is the following. (In this and all other statements in this paper, references to EIG trees such as $\texttt{tree}_i, \texttt{tree}_j$ and $\texttt{tree}_z$ are made only for nonfaulty processors; the assumption that $i$, $j$ or $z$ is nonfaulty will be implicit.)

LEMMA 6.2. *Let $F$ be an admissible function, and let $\sigma$ be a node of the safe subtree such that $0 < |\sigma| < t + 1 - \Delta$, and all nonfaulty processors that do not appear in $\sigma$ report the value $v$ for $\sigma$. Then*

*(a) If $v = \texttt{par}(|\sigma|)$ then $\sigma$ is closed in $\texttt{tree}_i$ by the end of round $|\sigma| + 1$ at the latest; while*

*(b) If $v \neq \texttt{par}(|\sigma|)$ then $\sigma$ is closed in $\texttt{tree}_i$ by the end of round $\min(|\sigma| + 2, t + 1 - \Delta)$ at the latest.*

*Moreover, if $\sigma$ is fixed in $\texttt{tree}_i$, then it is fixed to $v$.*

*Proof.* We prove the claim separately for different depths of $|\sigma|$.

(*i*) $t - \Delta > |\sigma| \geq 1$: We argue by cases depending on whether $v = \texttt{par}(|\sigma|)$.

(*a*) Assume $v = \texttt{par}(|\sigma|)$. If $\sigma$ is closed by the end of round $|\sigma|$ then $\sigma$ does not become fixed in $\texttt{tree}_i$ and we are done. Otherwise, $\sigma$ has at least $n - t - 1$ correct

children all of which store $v$ by the end of round $|\sigma|+1$. By rule Fx2 we thus obtain that $\sigma$ is fixed to $v$ at the end of round $|\sigma|+1$.

(*b*) Now assume $v \neq \mathtt{par}(|\sigma|)$. The node $\sigma$ cannot be fixed to $v \neq \mathtt{par}(|\sigma|)$ by rule Fx2, since this rule only allows fixing to $\mathtt{par}(|\sigma|)$. If $\sigma$ is closed by the end of round $|\sigma|+1$, then $\sigma$ does not become fixed in $\mathtt{tree}_i$ and we are done. We shall show that if $\sigma$ is not closed by the end of round $|\sigma|+1$ then $\sigma$ becomes fixed to $v$ by the end of round $|\sigma|+2$. Let $\sigma j$ be a correct child of $\sigma$. In particular, $\mathtt{tree}_i(\sigma j) = v$, and all correct children of $\sigma j$ store $v$ in $\mathtt{tree}_i$ as well. Notice that $\mathtt{par}(|\sigma j|) = v$ and $|\sigma j| \geq 2$. In addition, since $\sigma$ is a node of the safe subtree and $|\sigma| < t - \Delta$, the node $\sigma j$ has at least $n-t-2$ correct children, and they all store $v$. It follows that Fx1 does not apply to $\sigma j$, and $\sigma j$ becomes fixed to $v$ at the end of round $|\sigma|+2$ for all correct nodes $\sigma j$. Recall that there are at least $n-t-1$ such correct nodes $\sigma j$. Let $r = |\sigma|$, and recall that we are assuming that $F(r) \geq t - r + 2$.[3] Thus, in particular, $n - r - F(r) + 1 \leq n - r - t + r - 2 + 1 = n - t - 1$. It now follows by Fx3(b) that $\sigma$ is fixed to $v$ in $\mathtt{tree}_i$ by the end of round $|\sigma|+2$.

(*ii*) $|\sigma| = t - \Delta$: If $\sigma$ is closed at the end of round $|\sigma| = t - \Delta$, then $\sigma$ does not become fixed in $\mathtt{tree}_i$ and we are done. Otherwise, if $v = \mathtt{par}(|\sigma|)$ then $\sigma$ becomes fixed by rule Fx2 to $v$ as in the case of $t - \Delta > |\sigma|$ described above. If $v \neq \mathtt{par}(|\sigma|)$ then all of $\sigma$'s children are fixed in $\mathtt{tree}_i$ at the end of round $|\sigma|+1$. Moreover, at least $n-t-1$ correct children $\sigma j$ of $\sigma$ are fixed to $v$. As in the case of $|\sigma| > t - \Delta$ and $v \neq \mathtt{par}(|\sigma|)$ we have that $\sigma$ is fixed to $v$ in $\mathtt{tree}_i$ by rule Fx3(b).

□

Lemma 6.2 immediately provides us with a number of useful corollaries. Essentially, Lemma 6.2 implies that nodes corresponding to initially disabled processors and righteous nodes are guaranteed to close quickly given our fixing rules.

COROLLARY 6.3. *Let $F$ be admissible, and let $\sigma = \tau z$ be a node of the safe subtree such that $|\sigma| < t + 1 - \Delta$ and $z$ is disabled by the end of round $|\sigma|$. Then $\sigma$ is closed in $\mathtt{tree}_i$ by the end of round $|\sigma|+1$ at the latest. Moreover, if $\sigma$ becomes fixed to a value $v$ in $\mathtt{tree}_i$, then $v = \mathtt{par}(|\sigma|) = \mathtt{right}^F(|\sigma|)$.*

*Proof.* Since $|\sigma| < t + 1 - \Delta$, rule Fx1 does not apply to fixing $\sigma$. If $\sigma$ is closed in $\mathtt{tree}_i$ by the end of round $|\sigma|$, we are done. Otherwise, since we have that $z$ is disabled by the end of round $|\sigma|$ and $\sigma = \tau z$, all nonfaulty processors $j$ issue a $\mathtt{mask}(j, z)$ report by round $|\sigma|+1$ at the latest. As a result, all nonfaulty processors are considered to be reporting $v = \mathtt{par}(|\sigma|)$ for $\sigma = \tau z$. Lemma 6.2 now implies that $\sigma$ is closed in $\mathtt{tree}_i$ by the end of round $|\sigma|+1$, and, if it is fixed in $\mathtt{tree}_i$, it is fixed to $v = \mathtt{par}(|\sigma|)$. Since at least $n - t > n - t - 1 \geq F(|\sigma|)$ righteous children of $\tau z$ in $\mathtt{tree}_i$ store the value $\mathtt{par}(|\sigma|)$, we obtain that $\mathtt{par}(|\sigma|) = \mathtt{right}^F(\sigma)$ and we are done. □

In particular, Corollary 6.3 implies that all nodes corresponding to initially disabled processors become closed within one round, and can fix only to their masked value. A similar situation holds with respect to righteous nodes:

COROLLARY 6.4. *Let $F$ be admissible, and let $\sigma$ be a safe node (and hence a leaf of the safe subtree). Then $\sigma$ is closed in $\mathtt{tree}_i$ by the end of round $\min(|\sigma|+2, t+1-\Delta)$. Moreover, if $\sigma$ becomes fixed to a value $v$ in $\mathtt{tree}_i$, then $v = \mathtt{tree}_i(\sigma) = \mathtt{right}^F(\sigma)$.*

*Proof.* First notice that if $|\sigma| = t + 1 - \Delta$ and $\sigma$ is not closed by the end of round $t - \Delta$, then $\sigma$ is fixed in $\mathtt{tree}_i$ by rule Fx1 to $\mathtt{tree}_i(\sigma) = \mathtt{right}^F(\sigma)$ and we are done. If $|\sigma| = t + 1 - \Delta$ and $\sigma$ is closed in $\mathtt{tree}_i$ by the end of round $t - \Delta$, then $\sigma$ is not fixed in $\mathtt{tree}_i$ and we are done. Otherwise, we have that $|\sigma| < t + 1 - \Delta$.

[3] This is where we use the lower bound specified in the definition of admissible functions.

Let $\sigma = \tau z$, and assume that $\sigma$ is not closed by the end of round $|\sigma|$. If $\sigma$ is safe because $z \in \mathcal{D}$, then all processors report $\mathtt{par}(|\sigma|)$ for $\sigma$ in round $|\sigma|+1$ and we are done by Lemma 6.2. We may thus assume that $\sigma$ is righteous, then every nonfaulty processor $j$ reports $\mathtt{tree}_j(\sigma)$ for $\sigma$. Since we are assuming that $z$ is nonfaulty, then, by soundness of the fault detection module and the Recording condition, every nonfaulty processor $j$ would store the value that $z$ reports for $\tau$ in $\mathtt{tree}_j(\sigma) = \mathtt{tree}_j(\tau z)$. Moreover, a nonfaulty processor $z$ would report the same value for $\tau$ to all nonfaulty processors $j$. Thus, we have that $\mathtt{tree}_i(\sigma) = \mathtt{tree}_j(\sigma)$ for every nonfaulty processor $j$, and we obtain that all nonfaulty processors report the value $\mathtt{tree}_i(\sigma) = \mathtt{right}^F(\sigma)$ for $\sigma$. The claim now follows by Lemma 6.2. □

An immediate consequence of Corollaries 6.3 and 6.4 is:

LEMMA 6.5. *A node $\sigma$ that is not closed in $\mathtt{tree}_i$ by the end of round $|\sigma|+1$ is in the safe subtree.*

*Proof.* Corollaries 6.3 and 6.4 imply that every node $\tau$ on the safe cut $C_s$ is closed by the end of round $|\tau|+2$. Since these nodes form a cut in the EIG tree, every node $\sigma$ that is not in the safe subtree has an ancestor $\tau$ in $C_s$; moreover, $|\sigma|+1 \geq |\tau|+2$. It follows that a node $\sigma$ that is not in the safe subtree must be closed by the end of round $|\sigma|+1$ at the latest, and the claim follows. □

The fixing rule Fx3 has the property that once all children of a node are fixed, so is the node itself. As a result, we obtain:

LEMMA 6.6. *For each nonfaulty $i$, every node $\sigma$ of the safe subtree is closed in $\mathtt{tree}_i$ by the end of round $t+1-\Delta$.*

*Proof.* We prove the claim by induction on $h_s(\sigma)$ for nodes $\sigma$ of the safe subtree. Recall that we have defined $h_s(\sigma)$ to be the height of $\sigma$ in the safe subtree. In particular, we clearly have that $h_s(\sigma) \leq t+1-\Delta$ for every node $\sigma$ of the safe subtree. If $h_s(\sigma) = 0$, then $\sigma$ is a safe node, by definition of $h_s$. The claim now follows from Corollary 6.4. Now assume $h_s(\sigma) = k > 0$, and assume the claim holds for all nodes $\tau$ satisfying $h_s(\tau) < k$. In particular, every child $\sigma j$ of $\sigma$ is in the safe subtree and satisfies $h_s(\sigma j) < h_s(\sigma) = k$. Thus, the inductive assumption implies that all of $\sigma$'s children are closed by the end of round $t+1-\Delta$. If this is because $\sigma$ or one of its ancestors are fixed, we are done. Otherwise, all of $\sigma$'s children are fixed by the end of round $t+1-\Delta$. Recall that the number of $\sigma$'s children is $n-|\sigma|$. If rule Fx3(a) does not apply, then fewer than $t+2-|\sigma|$ of the children of $\sigma$ are fixed to the value $\mathtt{par}(|\sigma|)$. It then follows that at least $n-|\sigma|-(t+1-|\sigma|) = n-t-1$ of the children of $\sigma$ are fixed to $1-\mathtt{par}(|\sigma|)$, so that $\sigma$ is fixed to $1-\mathtt{par}(|\sigma|)$ in $\mathtt{tree}_i$ by rule Fx3(b). In either case we obtain that $\sigma$ must also be fixed in $\mathtt{tree}_i$, and we are done. □

A crucial property of our fixing rules is that a node $\sigma$ can be fixed to at most one value in $\mathtt{tree}_i$, as we now prove:

LEMMA 6.7. *If a node $\sigma$ of the safe subtree is fixed to value $v$ in $\mathtt{tree}_i$, then $\sigma$ is not fixed to $1-v$ in $\mathtt{tree}_i$.*

*Proof.* We prove the claim by induction on $h_s(\sigma)$. The case $h_s(\sigma) = 0$ follows from Corollary 6.4. Assume that $h_s(\sigma) \geq 1$ and that the claim holds for all nodes $\tau$ of the safe subtree with $h_s(\tau) < h_s(\sigma)$. Hence, in particular, rule Fx1 does not apply to $\sigma$, and the claim holds for $\sigma$'s children in $\mathtt{tree}_i$. Assume $\sigma$ is fixed by Fx2. The only value to which $\sigma$ can be fixed by Fx2 is $\mathtt{par}(|\sigma|)$. Moreover, by the definition of these rules, if Fx2 applies to $\sigma$ in $\mathtt{tree}_i$ then neither Fx1 nor Fx3 do. It follows that if $\sigma$ is fixed by rule Fx2, then it is fixed to a unique value. Finally, assume $\sigma$ becomes fixed in $\mathtt{tree}_i$ by rule Fx3. In particular, Fx1 and Fx2 do not apply in the case of $\sigma$. The node $\sigma$ has exactly $n-|\sigma|$ children $\sigma j$. By definition, $h_s(\sigma j) < h_s(\sigma)$. Thus, by the

inductive hypothesis, each of these children is fixed to at most one value. The total number of fixed children of $\sigma$ that would be necessary for both Fx3(a) and Fx3(b) to apply is $t+2-|\sigma|+n-t-1 = n-|\sigma|+1$, which is more than the number of children of $\sigma$. It follows that only one of these rules can apply, so that $\sigma$ can be fixed to at most one value in $\texttt{tree}_i$. □

We are now ready to prove that nodes of the safe subtree can become fixed only to their $\texttt{right}^F$ values.

THEOREM 6.8. *If $F$ is admissible and a node $\sigma$ of the safe subtree is fixed to value $v$ in $\texttt{tree}_i$, then $v = \texttt{right}^F(\sigma)$.*

*Proof.* We prove the claim by induction on $h_s(\sigma)$. The case of $h_s(\sigma) = 0$ follows directly from Corollary 6.4.

Assume that $h_s(\sigma) > 0$ and the claim holds for all children $\sigma j$ of $\sigma$. (By definition of $h_s$ we have that $h_s(\sigma) > h_s(\sigma j) \geq 0$.) Notice that the rule Fx1 cannot apply to $\sigma$ since Fx1 deals with nodes $\tau$ satisfying $|\tau| = t+1-\Delta$, and such a node is in the safe subtree only if it is a safe node, in which case $h_s(\tau) = 0$. Thus, $h_s(\sigma) > 0$, and only rules Fx2 and Fx3 can apply for fixing $\sigma$. Since $\sigma$ is an internal node of the safe subtree, $\sigma$ and all of its ancestors are incorrect nodes. As a result, at least $n-t$ of $\sigma$'s children are correct, while at most $t-|\sigma|$ of them are faulty. Moreover, every correct child of $\sigma$ is righteous. We now consider the ways in which $\sigma$ can become fixed due to Fx2 and Fx3.

(i) Assume that $|\sigma| \geq 2$ and $\sigma$ becomes fixed in $\texttt{tree}_i$ to value $v = \texttt{par}(\sigma)$ by rule Fx2. The definition of Fx2 implies that $\texttt{tree}_i(\sigma j) = v$ for at least $n-t-2$ nodes $\sigma j$. Since at most $t-|\sigma|$ of these may be incorrect, we obtain that at least $n-t-2-t+|\sigma| = n-2t+|\sigma|-2 \geq t+|\sigma|-1$ of these nodes are correct. For each such correct child $\sigma j$ we have that $\texttt{right}^F(\sigma j) = v$. Since $F$ is admissible, we have that $F(|\sigma|) \leq t+|\sigma|-1$, and it follows from the definition of $\texttt{right}^F$ that $\texttt{right}^F(\sigma) = v$.[4] A similar argument applies for the cases of $|\sigma| \leq 1$. In these cases, $F(|\sigma|) = t+1$, at least $n-t-|\sigma|$ of the children are fixed to $v$, and at most $t-|\sigma|$ of the children are incorrect. In both cases $n-t-|\sigma|-t+|\sigma| = n-2t \geq t+1 = F(|\sigma|)$ and we are done.

(ii) If $\sigma$ is fixed in $\texttt{tree}_i$ to $v$ by rule Fx3(a), then $\texttt{par}(|\sigma|) = v$ and we have by the inductive hypothesis for the $F(|\sigma|) \geq t+2-|\sigma|$ nodes of the form $\sigma j$ that are fixed to $v$ in $\texttt{tree}_i$ that $\texttt{right}^F(\sigma j) = v$. Thus, by the definition of $\texttt{right}^F$ we have that $\texttt{right}^F(\sigma) = v$ as well. An analogous argument works for fixing based on Fx3(b).

□

COROLLARY 6.9. *The root $\lambda$ of $\texttt{tree}_i$ is fixed to value $\texttt{right}^F(\lambda)$ by the end of round $t+1-\Delta$.*

*Proof.* Lemma 6.6 implies that $\lambda$ is closed in $\texttt{tree}_i$ by the end of round $t+1-\Delta$. Since $\lambda$ has no ancestors, it must be fixed at that time. Theorem 6.8 implies that $\lambda$ is fixed to $\texttt{right}^F(\lambda)$. □

THEOREM 6.10. *For any sound fault-detection module and admissible function $F$, the $\Delta$-EIG protocol satisfies the Decision, Agreement, and Validity properties.*

*Proof.* Decision follows immediately from Corollary 6.9 and Lemma 6.7. Recall from our discussion of $\texttt{right}^F$ that if $\sigma$ is in the righteous subtree, then the value of $\texttt{right}^F(\sigma)$ is independent of the tree in which it is computed. Corollary 6.9 implies that every nonfaulty processor decides on $\texttt{right}^F(\lambda)$, and since $\lambda$ is in the righteous subtree, we obtain Agreement. We now argue why Validity holds. All correct children

[4] This is where we use the upper bound specified in the definition of admissible functions.

of $\lambda$ are righteous. If all nonfaulty processors $j$ have the same initial value $v_j = v$, then $\texttt{right}^F(\sigma) = v$ for all $n - t$ righteous children $\sigma$ of $\lambda$. By definition of $\texttt{right}^F$, this implies that $\texttt{right}^F(\lambda) = v$, and since $\texttt{right}^F(\lambda)$ is the value decided on, we obtain Validity. □

**6.3. Fault detection in $\Delta$-EIG.** We now turn to describing the fault discovery rules we shall use in the instances of $\Delta$-agreement for the purposes of our final protocol. Notice that all of the results regarding fixing that we have seen above depend only on the Masking and Soundness rules, which state that initially detected failures must be masked to the favored value, and nonfaulty processors are not masked by other nonfaulty processors. Thus, we have a considerable amount of freedom in introducing fault discovery rules without affecting the correctness of the protocol.

We find it convenient to consider the notion of a node $\sigma$ being *committed to value $v$ in* $\texttt{tree}_i$. Intuitively, $\sigma = \tau z$ will be committed to $v$ in $\texttt{tree}_i$ only if $i$ has a proof that at least one nonfaulty processor either has received a report of $v$ for $\tau$ from $z$, or is masking $z$. Formally, we say that a node $\sigma \neq \lambda$ is committed to $v$ in $\texttt{tree}_i$ if one of the following holds:

C1: $\texttt{tree}_i(\sigma) = v$;

C2: $\texttt{tree}_i(\sigma j) = v$ for at least $\min(t + 1, t + 3 - |\sigma|)$ nodes $\sigma j$;

C3: $\sigma$ is not closed in $\texttt{tree}_i$ at the end of round $|\sigma| + 1$ and $\sigma$ is not fixed to $1 - v$ at the end of round $|\sigma| + 2$.

As in the case of fixing, a naive linear-time computation based on C1, C2 and C3 is easily seen to suffice for determining all of the commitments of nodes to values in a given EIG tree. We remark that these rules will only be applied to nodes of the safe subtree. For such nodes, the bound of C2 guarantees that one of the children $\sigma j$ with $\texttt{tree}_i(\sigma j) = v$ is correct.

The main use we have for the notion of commitment is captured in the following lemma:

LEMMA 6.11. *Let $\tau$ be a node of the safe subtree. If a child $\tau z$ of $\tau$ is committed to $v$ in $\texttt{tree}_i$ by the end of round $r$, then, for at least one nonfaulty processor $j$, either $z \in \mathcal{F}_j(|\tau z|)$ or $j$ received a report of $v$ from $z$ for $\tau$.*

*Proof.* If $\tau z$ is committed to $v$ in $\texttt{tree}_i$ by C1, then the claim holds trivially for $j = i$, since $\texttt{tree}_i(\tau z) = v$ only if either $i$ received a report of $v$ for $\tau$ from $z$, or $v = \texttt{par}(|\tau|)$ and $i$ is masking $z$ in round $|\tau z|$. Assume that $\tau z$ is committed to $v$ in $\texttt{tree}_i$ by C2. Since $\tau$ is a node of the safe subtree, all, except possibly for the last, members of the sequence $\tau$ are faulty processors. It follows that the number of incorrect children of $\tau z$ is at most $t - |\tau| + 1$ $(= t + 2 - |\tau z|)$. We thus obtain that if C2 applies, then at least one of the children $\tau z j$ of $\tau z$ with $\texttt{tree}_i(\tau z j) = v$ must be a correct node. But $\texttt{tree}_i(\tau z j) = v$ for a correct node $\tau z j$ only if either $z \in \mathcal{F}_j(|\tau z|)$ (in which case $j$ sends a $\texttt{mask}(j, z)$ message no later than in round $|\tau z| + 1$), or if $j$ received a report of $v$ for $\tau$ from $z$. We are left with the case of commitment to $v$ due to C3. First notice that if $\tau z$ is righteous and committed to $v$ in $\texttt{tree}_i$ by C3, then it is already committed to $v$ in $\texttt{tree}_i$ by C2. This is because if $\tau z$ is not closed in $\texttt{tree}_i$ at the end of round $|\tau z| + 1$, then it is committed in $\texttt{tree}_i$ to value $w = \texttt{tree}_i(\tau z)$, and by Corollary 6.4, it will not not be committed to $1 - w$ by C3. Thus, we may assume that $\tau z$ is not righteous. In particular, it has at least $n - t$ righteous children. It suffices to show that at least one nonfaulty processor $j$ reports $v$ for $\tau z$. Assume not. Then all nonfaulty processors report $1 - v$ for $\tau z$. It now follows by Lemma 6.2 that $\tau z$ must be fixed to $1 - v$ by the end of round $|\tau z| + 2$ if it was not closed by the end of round $|\tau z| + 1$. This contradicts the assumption that $\tau z$ is committed to $v$

in $\texttt{tree}_i$ by C3. □

LEMMA 6.12. *Assume that $t \geq 3$. If a node $\sigma \neq \lambda$ of the safe subtree is ever fixed to value $v$ in $\texttt{tree}_i$, then $\sigma$ is committed to $v$ in $\texttt{tree}_i$ by the end of round $|\sigma| + 2$.*

*Proof.* Assume $\sigma$ is fixed to $v$ in $\texttt{tree}_i$ by the end of round $|\sigma| + 2$. If $\sigma$ is fixed by rule Fx1, then by C1 it is also committed to $v$ in $\texttt{tree}_i$. If it is fixed by Fx2, then it is committed to $v$ by C2, since Fx2 implies that $\texttt{tree}_i(\sigma j) = v$ for at least $n - t - 2$ nodes $\sigma j$. Given that $t \geq 3$ and $|\sigma| \geq 1$, we have

$$n - t - 2 \geq 2t + 1 - 2 = 2t - 1 \geq t + 3 - 1 = t + 2 \geq t + 3 - |\sigma|.$$

Finally, assume that $\sigma$ is fixed to $v$ by Fx3. In particular, $\sigma$ is not closed in $\texttt{tree}_i$ by the end of round $|\sigma| + 1$, and it becomes fixed to $v$ no earlier than the end of round $|\sigma| + 2$. Lemma 6.7 implies that $\sigma$ can be fixed to at most one value in $\texttt{tree}_i$, so that $\sigma$ is not fixed to $1 - v$ at the end of round $|\sigma| + 2$, and by C3 it is committed to $v$ in $\texttt{tree}_i$ at that point. □

Notice that for every node $\sigma \neq \lambda$ of the safe subtree, there must be at least one value $v \in \{0, 1\}$ such that at least $t + 1$ (and hence $\geq t + 2 - |\sigma|$) nonfaulty processors report $v$ for $\sigma$. We say that $\sigma$ is then *publicly committed* to such a value $v$. If $\sigma$ is publicly committed to $v$, then $\sigma$ becomes committed by C2 to $v$ in $\texttt{tree}_i$ for all trees $\texttt{tree}_i$ in which values of children of $\sigma$ are stored. A variant of Lemma 6.12 that applies to public commitment and will be useful in the sequel is the following:

LEMMA 6.13. *Let $\sigma$ be a node of the safe subtree of depth $2 \leq |\sigma| \leq t - \Delta$, let $v = \texttt{par}(|\sigma|)$, and let $\sigma$ be fixed in $\texttt{tree}_i$ to $1 - v$ (the "disfavored" value). Then $\sigma$ is publicly committed to $1 - v$.*

*Proof.* By Theorem 6.8, a node $\sigma$ of the safe subtree can be fixed only to $\texttt{right}^F(\sigma)$. Given our definition of publicly committed, we have that a node must be publicly committed to at least one value among $0, 1$. However, if $\sigma$ were publicly committed to $v = \texttt{par}(|\sigma|)$, then by definition of $\texttt{right}^F$ we would have that $\texttt{right}^F(\sigma) = \texttt{par}(|\sigma|) \neq 1 - v$. It follows that $\sigma$ must be publicly committed to $1 - v$. □

We are now ready to present our fault detection rules. Intuitively, a nonfaulty processor $i$ discovers that a processor is faulty when there is "enough" evidence that the processor has sent conflicting values to other correct processors. This evidence may be gathered when the messages of the children of a node corresponding to the faulty processor are received. Formally, processor $i$ will detect $z$ at the end of round as being faulty if one of the following holds:

**FD0:** $z$ sends $i$ an ill-formatted message in round $r$.

**FD1:** By the end of round $r$, processor $i$ has received $\texttt{mask}(j, z)$ reports from at least $t + 1$ distinct processors $j$.

**FD2:** By the end of round $r$, some node $\tau z$ that was not closed in $\texttt{tree}_i$ by the end of round $|\tau z|$ is committed both to 0 and to 1 in $\texttt{tree}_i$.

**FD3:** By the end of round $r$, for some node $\tau a z$ and value $v$ such that (a) $r = |\tau a z| + 1$, and (b) $\tau a z$ is not closed in $\texttt{tree}_i$ by the end of round $r$; we have that

(i) $\tau a z$ is committed to $v$ in $\texttt{tree}_i$;

(ii) at least $2(t + 1 - |\tau a|) + 1$ of the nodes $\tau a j$ are committed to $1 - v$ in $\texttt{tree}_i$ by the end of round $r$; and

(iii) $z$ does not mask $a$ in round $|\tau a z| + 1$. (Namely, $z$ did not send a $\texttt{mask}(z, a)$ report to $i$ in the first $r$ rounds.)

The motivation for the first three rules FD0–FD2 is fairly intuitive and straightforward. Similar rules have appeared in earlier work in the literature. The fourth rule,

FD3, is of a new type. It is tailor-made for handling a specific type of corruption, called cross corruption, that we will consider in detail in Section 7. Intuitively, FD3 can be thought of as detecting a *crime of ommission*. It applies when the detecting processor $i$ has a proof that, had $z$ been nonfaulty, then $z$ would have detected another processor $a$ as being faulty due to FD2 in round $r-1$. As a result, $z$ should have issued a $\texttt{mask}(z,a)$ report no later than in round $r$. By rule FD3, $i$ discovers $z$ as being faulty once $z$ fails to issue a $\texttt{mask}(z,a)$ in time. Figure 2 illustrates this scenario.

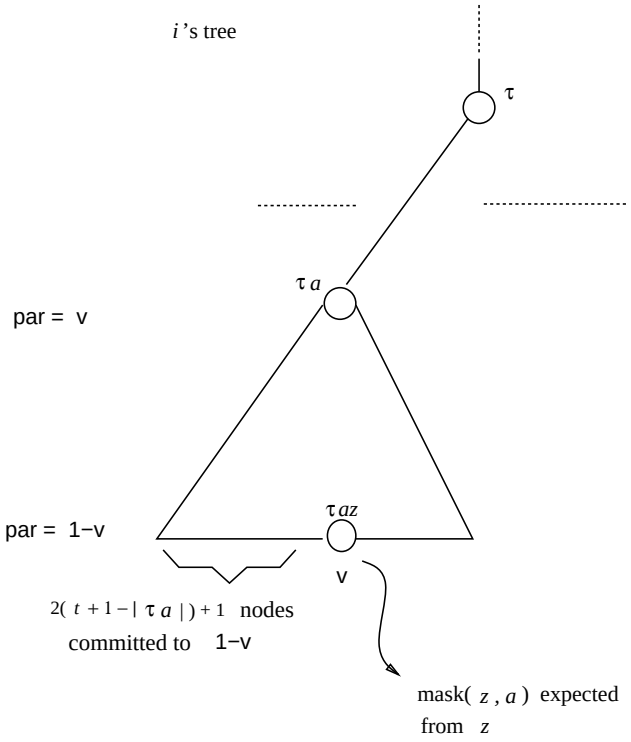


FIG. 2. *Discovering a fault using* FD3.

LEMMA 6.14. *The rules* FD0–FD3 *are sound. That is, if the fault detection module is sound for all nonfaulty processors through the end of round* $r-1$*, then every processor that is added to* $\mathcal{F}_i(r)$ *according to one of these rules is faulty.*

*Proof.* The soundness of FD0 is trivial, since a processor that sends an ill-formatted message is deviating from the protocol, and hence is faulty. If rule FD1 applies, then at least one of the $t+1$ processors that report to $i$ that they are masking $z$ must be nonfaulty. The soundness of the rule now follows from our assumption about the soundness of the fault detection module in the first $r-1$ rounds. We now turn to FD2. Notice that this rule cannot apply with respect to a node $\tau z$ before the end of round $|\tau z|+1$, since only one commitment value can be obtained in $\texttt{tree}_i$ before values for the children of $\tau z$ are stored. By Lemma 6.5, if $\tau z$ is not closed at the end of round $|\tau z|$, then $\tau$ is a node of the safe subtree. Since $\tau z$ is committed both

to 0 and to 1, Lemma 6.11 implies that either at least one nonfaulty processor is masking $z$ in round $|\tau z|$, or $z$ sent conflicting reports of both 0 and 1 for $\tau$ to two different nonfaulty processors. Having assumed that the fault detection module was sound in the first $r-1$ rounds we obtain that, in either case, $z$ must be faulty, and hence FD2 is sound.

Finally, let us consider FD3. Roughly speaking, the soundness of FD3 is based on FD2 and the fact that if (i) and (ii) hold, then $i$ can determine that $z$ must have had enough information for detecting $a$ as faulty using FD2 in round $|\tau az|$. If $z$ did not react accordingly, then it must be faulty. We now formalize this intuition. Assume that $z$ is nonfaulty, and conditions (i) and (ii) apply. Moreover, assume that $z$ does not issue a $\texttt{mask}(z,a)$ report by the end of round $|\tau az|$. Since $\tau az$ is not closed by the end of round $|\tau az|+1$, then by Lemma 6.5 the node $\tau az$ is in the safe subtree, and hence all of the processors in the sequence $\tau a$ are faulty. It follows that at most $t-|\tau a|$ of the nodes $\tau aj$ that are committed to $1-v$ in $\texttt{tree}_i$ by the end of round $|\tau az|+1$ can be incorrect. $t-|\tau a|$ can be incorrect. Since their total number is, by assumption, at least $2(t+1-|\tau a|)+1$, it follows that at least

$$2(t+1-|\tau a|)+1-(t-|\tau a|)=t+3-|\tau a|$$

of these nodes are correct, and in particular must appear in $\texttt{tree}_z$. It thus follows by rule C2 that $\tau a$ must be committed to $1-v$ in $\texttt{tree}_z$ by the end of round $|\tau az|$. However, since $\tau az$ is committed to $v$ in $\texttt{tree}_i$, (and $z$ is nonfaulty) it must be the case that $\tau a$ is committed to $v$ in $\texttt{tree}_z$ as well by the end of round $|\tau az|$. Thus, by rule FD2 we obtain that $z$ must detect $a$ as faulty in round $|\tau az|$. Moreover, by the Masking behavior rule, in round $|\tau az|+1$, processor $z$ must report that it is masking $a$, if it has not done so in an earlier round. If $z$ fails to do so, then $z$ must be faulty as determined by FD3. □

Given these fault discovery rules, we can now turn to study the conditions under which nodes can be corrupted in instances of $\Delta$-EIG. In addition, we shall be interested in the relationship between the corrupted nodes and the size of the EIG tree constructed.

**6.4. Corrupting nodes in $\Delta$-EIG.** Formally, in an execution of the $\Delta$-EIG protocol we define a node $\sigma$ to be *corrupted in* $\texttt{tree}_i$ if $\sigma$ is not closed in $\texttt{tree}_i$ by the end of round $|\sigma|+2$.

A node $\sigma$ is said to be *universally corrupted* if it is corrupted in $\texttt{tree}_i$ for all nonfaulty processors $i$. The following three lemmas show that, in order to cause lasting trouble, a node must be universally corrupted.

LEMMA 6.15. *Let $i$ and $j$ be nonfaulty processors, and let $\sigma$ be a node of the safe subtree. If $\sigma$ is fixed in $\texttt{tree}_i$ by rule Fx2, then $\sigma$ is closed in $\texttt{tree}_j$ by the end of round $|\sigma|+3$.*

*Proof.* The claim follows immediately for righteous nodes $\sigma$ by Corollary 6.4. We shall henceforth consider the case in which $\sigma$ is not righteous, so that at most $t-|\sigma|$ of its children are faulty. We prove the claim for $|\sigma|\geq 2$; the modifications for the cases $|\sigma|=0$ and $|\sigma|=1$ are simple and left for the reader. Assume that $\sigma$ is fixed in $\texttt{tree}_i$ by rule Fx2, and that $\sigma$ is not closed in $\texttt{tree}_j$ by the end of round $|\sigma|+2$. Since $\sigma$ is fixed to $v$ in $\texttt{tree}_i$ by Fx2 in round $|\sigma|+1$, we have that at least $n-t-2-(t-|\sigma|)=n-2t+|\sigma|-2\geq t+|\sigma|-1$ of $\sigma$'s children in $\texttt{tree}_j$ are righteous children that store $v$. If $\sigma$ is not closed in $\texttt{tree}_j$ beforehand, then Corollary 6.4 implies that these nodes will all be fixed to $v$ in $\texttt{tree}_j$ by the end of round $|\sigma|+3$.

Since $F(|\sigma|) \leq t + |\sigma| - 1$ we have that $\sigma$ becomes fixed to $v$ in $\texttt{tree}_j$ by Fx3(a) at the end of round $|\sigma| + 3$. □

As a consequence of Lemma 6.15, a straightforward induction yields:

COROLLARY 6.16. *Let $i$ and $j$ be nonfaulty processors, and let $\sigma$ be a node of the safe subtree. If $\sigma$ is closed in $\texttt{tree}_i$ by the end of round $r$, then it is closed in $\texttt{tree}_j$ by the end of round $r + 2$.*

COROLLARY 6.17. *Let $i$ be a nonfaulty processor, and let $\tau$ be a node that is not universally corrupted. Then $\tau$ is closed in $\texttt{tree}_i$ by the end of round $|\tau| + 4$.*

*Proof.* By definition of corruption, a node $\tau$ that is not universally corrupted must be closed in $\texttt{tree}_j$ for some nonfaulty $j$ by the end of round $|\tau| + 2$. By Corollary 6.16 we have that $\tau$ is closed in $\texttt{tree}_i$ by the end of round $|\tau| + 4$. □

As in the case of $n > 4t$, there is a close relationship between corrupted nodes and failure detection. Indeed, an immediate consequence of the fault discovery rule FD2 and the commitment rule C3 is the fact that a processor that corrupts a node in $\texttt{tree}_i$ is discovered by $i$ as being faulty. Formally, we have:

LEMMA 6.18. *If a node $\tau z$ is corrupted in $\texttt{tree}_i$, then $z \in \mathcal{F}_i(|\tau z| + 2)$.*

Given Lemmas 6.2 and 6.18, we obtain the following relationship between universal corruption and disabled processors:

COROLLARY 6.19. *If a node $\tau z$ is universally corrupted, then processor $z$ is disabled from the end of round $|\tau z| + 2$ on.*

Since corruption is determined within at most two rounds, Corollary 6.19 implies that a faulty processor can universally corrupt nodes in at most two different rounds. Thus, our situation resembles that of the $n > 4t$ case with majority, except that now the faulty processors are able to corrupt nodes in two consecutive rounds in some cases. This will force us to consider the possibility of cross-corruption in Section 7.

**6.4.1. Waste.** One consequence of Corollary 6.17 is that if no node $\sigma$ of depth $r$ is universally corrupted, then all such nodes are closed by the end of round $r + 4$. Moreover, it is easy to see that if all nodes of depth $r$ are closed, then so are all of their ancestors, including the root. We thus have

LEMMA 6.20. *If no node $\sigma$ of depth $|\sigma| = r$ is universally corrupted, then the root $\lambda$ is closed in all processors' trees by the end of round $r + 4$.*

We say that a processor $z$ *universally corrupts a node $\tau z$ at depth $r$* if the node $\tau z$ is universally corrupted, and $|\tau z| = r$. Notice that whether $z$ universally corrupts $\tau z$ might depend on events that take place after round $r = |\tau z|$, such as what values other faulty processors report for $\tau z$, and who is masking $z$ in round $r+1$. Intuitively, we can figure out whether $z$ universally corrupted $\tau z$ only in round $r+2$. Corollary 6.19 states that at most two rounds after a processor universally corrupts a node, this processor is disabled. Corollary 6.3 implies that a disabled processor cannot universally corrupt nodes. It follows that a processor can universally corrupt nodes in at most two (consecutive) rounds. However, it is not hard to see that a processor $z$ cannot be the only processor universally corrupting nodes in a pair of consecutive rounds $r$ and $r+1$:

LEMMA 6.21. *Assume that $z$ universally corrupts nodes at depth $r$. If there are universally corrupted nodes at depth $r + 1$, then there must be at least one processor $z' \neq z$ corrupting nodes either at depth $r$ or at depth $r + 1$.*

*Proof.* Assume that $\tau xy$ is a node of depth $|\tau xy| = r + 1$ that is universally corrupted. If no processor other than $z$ universally corrupts nodes at depth $r + 1$, we must have that $y = z$. By definition of the EIG tree, a node is a sequence of processor names without repetitions, and hence $x \neq z$. Since $\tau xy$ is universally corrupted, it is not closed in any processor's tree by the end of round $|\tau| + 4$. It follows that $\tau x$ is also

not closed in any processor's tree at that point. We conclude that $\tau x$ was not closed in any tree at the end of round $|\tau|+3 = |\tau x|+2$, and hence was universally corrupted. Thus, $x \neq z$ universally corrupted nodes at depth $r$, and the claim follows. □

Lemma 6.20 implies that in order to keep the trees from closing, at least one processor must universally corrupt nodes at every depth. Moreover, Lemma 6.21 implies that at least two different processors must universally corrupt nodes in consecutive rounds. Finally, by Corollary 6.19, we know that two rounds after a processor universally corrupts nodes, this processor is disabled. It follows that, roughly speaking, in order to keep the root from closing, at least one processor per round must become disabled. We now make a few definitions that will allow us to make this intuition into a precise statement, and will later help us in the analysis of monitor voting.

Let us denote by $\mathcal{D}(r)$ the set of processors that are disabled at the end of round $r$. We define the *deficit* at $r$, denoted $\texttt{deficit}(r)$, to be the number of processors that universally corrupt nodes at depth $r-1$, but are not disabled by the end of round $r$. Recall that every processor that universally corrupts a node at depth $r-1$ is detected by all nonfaulty processors as faulty (and hence disabled) by the end of round $r+1$. We thus have $\#\mathcal{D}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r)$. We are almost ready to define the *waste* at $r$. Roughly speaking, the term *waste* comes from the idea that the nonfaulty processors are playing against an adversary.[5] This adversary is trying to enlarge the size of processors' trees without spending more than one disabled processor per round. The waste measures the extent by which the adversary has exceeded this allowance. Intuitively, the waste should be a measure, stated in terms of the number of disabled processors and the deficit, that will have the following properties:

1. As long as at least one node is universally corrupted in every round, the waste should be nondecreasing;

2. an appropriate form of monitor voting will guarantee that if the waste exceeds a certain constant threshold, then the agreement process will be halted; and

3. as long as the waste does not exceed the threshold of (2), then the number of universally corrupted nodes in the tree is polynomial.

To obtain this, we define the *waste* at the end of round $r$, denoted by $\texttt{Waste}(r)$, as follows:

$$\texttt{Waste}(r) \;=\; \#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r).$$

The last term, $\texttt{correction}(r)$, is technically needed in order to compensate for cases in which the processors that form the deficit are able to corrupt nodes at depth $r$ in addition to round $r-1$. In this case, we want to account one of these processors to the following round. Formally, we define $\texttt{correction}(r)$ as follows:

$$\texttt{correction}(r) = \begin{cases} 0 & \text{if } \texttt{deficit}(r) = 0; \\ 0 & \text{if } \texttt{deficit}(r) = 1 \text{ and only one processor universally} \\ & \qquad\qquad \text{corrupts nodes at depth } r-1; \quad \text{and} \\ 1 & \text{otherwise.} \end{cases}$$

In other words, the correction is 1 if either $\texttt{deficit}(r) \geq 2$, or if $\texttt{deficit}(r) = 1$, provided there is at least one processor universally corrupting nodes at depth $r-1$ in addition to the processor forming the deficit at $r$. In either case, we deduct a cost of 1 for the fact that some processor in the deficit had the option of causing harm at depth $r$ as well as at depth $r-1$. The following lemma makes this claim precise.

[5] The notion of *waste* used here is close in spirit to, though technically quite different from, a similar notion introduced in the work of Dwork and Moses [12].

LEMMA 6.22. *If* $\texttt{correction}(r) = 0$ *then no processor universally corrupts nodes both at depth* $r-1$ *and at depth* $r$.

*Proof.* We consider the two cases in which $\texttt{correction}(r) = 0$. For the first case, if $\texttt{deficit}(r) = 0$, then all processors that universally corrupt nodes in round $r-1$ are disabled by the end of round $r$, and hence cannot corrupt nodes at depth $r$. For the other case, assume that $\texttt{deficit}(r) = 1$ and let $z$ be the processor forming the deficit at $r$. By definition, if $\texttt{correction}(r) = 0$ then $z$ is the only processor that universally corrupts nodes at depth $r-1$. By Lemma 6.21, $z$ cannot also universally corrupt nodes at depth $r$. □

We can now prove:

THEOREM 6.23. *Let* $r \geq 2$. *If* $\texttt{Waste}(r+1) < \texttt{Waste}(r)$ *then no node of depth* $r$ *is universally corrupted.*

*Proof.* We prove the contrapositive statement: Assume that there is at least one universally corrupted node at depth $r$, and we shall show that $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$. Recall from the discussion above that $\#\mathcal{D}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r)$. In addition, notice that, by definition of correction, we are guaranteed that $\texttt{deficit}(r') - \texttt{correction}(r') \geq 0$ for all $r'$, and in particular we will have that $\texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq 0$. We consider three cases:

(i) At least one processor universally corrupting nodes at depth $r-1$ also universally corrupts nodes at depth $r$. In this case, we have by Lemma 6.22 that $\texttt{correction}(r) = 1$. We thus have:

$$\begin{aligned}\texttt{Waste}(r+1) = \#\mathcal{D}(r+1) - (r+1) + \texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq \\ \#\mathcal{D}(r+1) - (r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) - r - 1 = \\ \#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r) = \texttt{Waste}(r).\end{aligned}$$

(ii) The assumption of case (i) does not hold, and at least one processor $z$ that universally corrupts nodes at depth $r$ is disabled by the end of round $r+1$. Clearly, $z \notin \mathcal{D}(r)$, and by assumption we have that $z$ is not one of the processors forming a deficit at $r$. Since $z$ is disabled by the end of round $r+1$, we have that $\#\mathcal{D}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) + 1$, and hence we are guaranteed that $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$ by:

$$\begin{aligned}\texttt{Waste}(r+1) \geq \#\mathcal{D}(r+1) - (r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) + 1 - (r+1) = \\ \#\mathcal{D}(r) + \texttt{deficit}(r) - r \geq \#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r) = \texttt{Waste}(r).\end{aligned}$$

(iii) The assumptions of cases (i) and (ii) do not hold. Thus, no processor universally corrupts nodes both at depth $r-1$ and at depth $r$, and no processor that universally corrupts nodes at depth $r$ is disabled by the end of round $r+1$. It follows that only processors forming the deficit at $r+1$ universally corrupt nodes at depth $r$. In this case, if $\texttt{deficit}(r+1) \geq 2$, then $\texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq 1$, and we have that that $\texttt{Waste}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) + 1 - (r+1)$ and hence $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$ as in the previous case. Finally, if $\texttt{deficit}(r+1) = 1$ then the assumptions imply that the processor forming the deficit at $r+1$ is the only processor universally corrupting nodes at depth $r$. By the definition of correction we thus have that $\texttt{correction}(r+1) = 0$, and we again obtain that $\texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq 1$, so that $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$ and we are done.

□

Theorem 6.23 will be instrumental in the correctness of our ultimate algorithm. We shall design the monitor voting scheme in such a way that once the waste becomes

large enough (which in our case will mean at least two), an appropriate monitor decision process will "fire", thereby causing the processors all to decide on a default value and halt.

In Section 7.3 we shall demonstrate how it is possible to stop interacting about a subtree of the EIG tree after it becomes closed. As a result, a fundamental parameter determining the size of the processors' trees will be the number of universally corrupted nodes in the tree. Theorem 6.23 allows us to formalize the idea that there is a close relationship between the waste of an execution and the number of universally corrupted nodes in the tree. Recall, for example, that if there is only one processor universally corrupting nodes at any given depth, then the total number of universally corrupted nodes is no greater than $t$. In order for this number to grow, it is necessary for there to be levels of the tree at which two or more processors universally corrupt nodes. Notice, however, that if three or more processors universally corrupt nodes at depth $r$, then Corollary 6.19 implies they all will be disabled by the end of round $r+2$, and as a result we would have that $\mathtt{Waste}(r+2) > \mathtt{Waste}(r)$. As we are going to start with a waste at $r = 2$ of at least $-1$, it will follow that after a constant number of such rounds, the monitors will detect a problem and stop the growth of the EIG tree.

The only situation in which the number of universally corrupted nodes can grow more than in a linear fashion, and the waste need not increase, is in the case of *cross corruption*. This is a situation in which two processors, say $a$ and $b$, universally corrupt nodes at depth $r$, and they continue to be the only processors to universally corrupt nodes at depth $r+1$. Specifically, if $a$ corrupted a node $\tau a$ at depth $r$ and $b$ corrupted $\tau b$, then at depth $r+1$ we will find $b$ universally corrupting $\tau ab$, while $a$ corrupts $\tau ba$. See Figure 3 for an illustration of this situation. In this fashion, it is possible to double the number of universally corrupted nodes of depth $r+1$ compared to the number of corrupted nodes of depth $r-1$, without increasing the waste.[6]

The next section is devoted to cross corruption. In particular, we shall devise an admissible resolve function that will restrict the number of times at which cross corruption need not increase the waste.

**7. Cross corruption.** As discussed in the last section, in order for the number of universally corrupted nodes to grow significantly without the waste growing at the same time, pairs of faulty processors need to cross-corrupt nodes in consecutive rounds. In this section, we perform a careful analysis of the conditions that must be met for cross-corruption to succeed. We then use this analysis to fine-tune the function $F$ we use for fixing nodes, to limit the number of times that a cross corruption can take place without the waste increasing. This analysis will yield essentially all of the necessary ingredients for our final protocol.

The first property of cross corruption we shall use is the fact that in cross corruption, each universally corrupted node $\tau a$ in the first of the two rounds has at most one universally corrupted child $\tau ab$. As a result, if the node $\tau a$ is not closed in a small number of rounds, then all of its children other than $\tau ab$ must be fixed in all trees. Moreover, since $\tau a$ is not closed after these nodes become fixed, the number of its children fixed to $\mathtt{par}(|\tau a|)$ and the number fixed to its complement are uniquely determined. More specifically, we have:

LEMMA 7.1. *In a* $\Delta$-EIG *protocol with admissible function* $F$, *let* $v = \mathtt{par}(|\tau a|)$ *and let* $\tau a$ *be a universally corrupted node whose only universally corrupted child*

[6] Indeed, Berman and Garay [5] have shown that for the resolve function of [2] for $n > 3t$, it is possible to construct an exponential tree this way, despite fault masking, early stopping, and monitor voting.

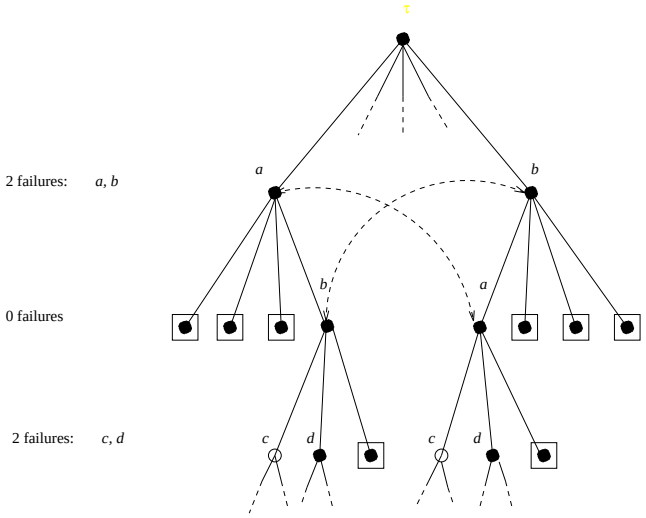


FIG. 3. *Cross corruption.*

*is $\tau ab$. For every nonfaulty processor $i$, if $\tau a$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau| + 6$, then all of its children other than $\tau ab$ are fixed in $\texttt{tree}_i$ at that point. Moreover, let $A$ denote the set of processors $x$ for which the node $\tau ax$ is fixed to $v$ in $\texttt{tree}_i$, and let $B$ denote the set of processors $y$ such that $\tau ay$ is fixed to $1-v$. Then $\#A = F(|\tau a|) - 1$ and $\#B = n - |\tau a| - F(|\tau a|)$.*

*Proof.* The scenario described in the statement of the lemma is depicted in Figure 4 near Lemma 7.7. Corollary 6.17 implies that a node $\tau ax$ that is not universally corrupted must be closed in all trees by the end of round $|\tau ax| + 4 = |\tau| + 6$. Since we are assuming that $\tau a$ is not closed in $\texttt{tree}_i$ at that point, it follows that every such node $\tau ax$ will necessarily be fixed in $\texttt{tree}_i$ by the end of round $|\tau| + 6$. Since $\tau ab$ is the only universally corrupted child of $\tau a$, we obtain that all other children of $\tau a$ must be fixed in $\texttt{tree}_i$ by the end of round $|\tau| + 6$. Let $A$ be set of processors $x$ such that the node $\tau ax$ is fixed to $v$ in $\texttt{tree}_i$ (by the end of round $|\tau| + 6$), and let $B$ the set of processors $y$ such that the node $\tau ay$ is fixed to $1 - v$. This is depicted in Figure 4. Since $\tau a$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau| + 6$ we are guaranteed that $\#A < F(|\tau a|)$ and $\#B < n - |\tau a| + 1 - F(|\tau a|)$. However, the fact that only one of $\tau a$'s children is universally corrupted implies that $\#A + \#B = n - |\tau a| - 1$. It follows that $\#A = F(|\tau a|) - 1$ and $\#B = n - |\tau a| - F(|\tau a|)$. □

Another observation regarding cross corruption is the following. Recall that every node of the safe subtree must be publicly committed either to 0 or to 1. If a node $\tau ac$ is publicly committed to $v$ and then becomes fixed to $1-v$, then by rule FD2 everybody will discover that $c$ is faulty by the end of round $|\tau ac| + 2 = |\tau| + 4$, and $c$ will become disabled by then. As a result, we obtain:

LEMMA 7.2. *Under the conditions and notation of Lemma 7.1, let $C \subseteq B$ consist of the processors $c$ such that $\tau ac$ is publicly committed to $v$. Then $C \subseteq (\mathcal{D}(|\tau| + 5) \setminus \mathcal{D}(|\tau| + 2))$.*

*Proof.* By definition of $C$, for every nonfaulty processor $j$ we have that each node $\tau ac$ with $c \in C$ is committed to $v$ in $\texttt{tree}_j$. In addition, since $C \subseteq B$, we have that $\tau ac$ is fixed to $\texttt{right}^F(\tau ac) = 1 - v$ in $\texttt{tree}_i$ by the end of round $|\tau| + 6$. As a result, we have by Theorem 6.8 and Lemma 6.7 that $\tau ac$ cannot become fixed

to $v$ in $\texttt{tree}_j$. Moreover, since, by assumption, $\tau a$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+6$, we have by Corollary 6.16 that $\tau a$ is not closed in $\texttt{tree}_j$ by the end of round $|\tau|+4=|\tau ac|+2$. Hence, $\tau ac$ can be closed in $\texttt{tree}_j$ at the end of round $|\tau ac|+2$ only if it is fixed (to $1-v$) in $\texttt{tree}_j$ at that point. If it is indeed fixed to $1-v$ in $\texttt{tree}_j$ at the end of round $|\tau|+4$, then $\tau ac$ is committed at that point to $1-v$ in $\texttt{tree}_j$ by Lemma 6.12. If not, then it is committed to $1-v$ at that point by C3. Since $\tau a$ is not closed in $\texttt{tree}_j$ by the end of round $|\tau|+4|$, we have that $\tau ac$ is not closed in $\texttt{tree}_j$ by the end of round $|\tau ac|$. Thus, in either case, rule FD2 implies that $j$ detects $c$ as faulty by the end of round $|\tau|+4$. Since this argument applies for every nonfaulty processor $j$, we have that $c$ is disabled at the end of round $|\tau|+4$. Notice, however, that $c$ could not have been disabled at the end of round $|\tau|+2$, for if it had been disabled at that point, then, by Lemma 6.2, $\tau ac$ would not be publicly committed to $v$: all nonfaulty processors would be reporting $1-v$ for $\tau ac$ by masking $c$. By definition of $B$, this contradicts the assumption that $c\in C\subseteq B$. We conclude that $C\subseteq(\mathcal{D}(|\tau|+4)\setminus\mathcal{D}(|\tau|+2))$. Since $\mathcal{D}(|\tau|+5)\supseteq\mathcal{D}(|\tau|+4)$, we obtain that $C\subseteq(\mathcal{D}(|\tau|+5)\setminus\mathcal{D}(|\tau|+2))$ and we are done. □

Lemma 7.2 implies that, in order to avoid having the waste grow in an instance of cross corruption, the size of the set $C$ must remain small. Specifically, as we shall see later on, it is necessary that $\#C\leq 2$. When this is the case, then the vast majority of nodes $\tau ay$ with $y\in B$ must be publicly committed to $1-v$. As a result, if $B$ is large enough, we can use the fault-detection rule FD3 to force the processors in $A$ to mask $a$ in round $|\tau|+3$. More specifically, we have:

LEMMA 7.3. *Under the conditions and notation of Lemma 7.1, let $B'\subseteq B$ consist of the processors $y'\in B$ such that the node $\tau ay'$ is publicly committed to $1-v$. Assume $\#B'\geq 2(t+1-|\tau a|)+1$. If the node $\tau ba$ is not closed in $\texttt{tree}_i$ at the end of round $|\tau|+7$, then every processor $x\in A$ for which $\tau bax$ is not fixed to $1-v$ in $\texttt{tree}_i$ by the end of round $|\tau|+7$ is disabled by the end of round $|\tau|+5$.*

*Proof.* Consider two cases:

(i) $\tau bax$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$: In this case, by Corollary 6.17 we have that $\tau bax$ is universally corrupted, and by Corollary 6.19 we obtain that $x$ is disabled by the end of round $|\tau bax|+2=|\tau|+5$.

(ii) $\tau bax$ is closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$: Since, by assumption, $\tau ba$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$, if $\tau bax$ is closed in $\texttt{tree}_i$ at that point, then it must be fixed in $\texttt{tree}_i$. Moreover, since we have assumed that $\tau bax$ is not fixed to $1-v$, it must be fixed to $v$ in $\texttt{tree}_i$ by the end of round $|\tau|+7$. As a result, for every nonfaulty processor $j$, the node $\tau bax$ does not become fixed to $1-v$ in $\texttt{tree}_j$. We now show that every nonfaulty processor $j$ must have detected $x$ as faulty no later than in round $|\tau|+5$.

Let $j$ be an arbitrary nonfaulty processor. Recall that every node $\tau ax$ with $x\in A$ is fixed in $\texttt{tree}_i$ to $v\neq\texttt{par}(|\tau ax|)$ by the end of round $|\tau|+6$. By Lemma 6.13 we thus have that every node $\tau ax$ with $x\in A$ is publicly committed to $v$. If $x$ does not send $j$ a $\texttt{mask}(x,a)$ message by the end of round $|\tau|+3$ then the conditions of the fault detection rule FD3 hold for $j$ with respect to $x$:

(i) $\tau ax$ is committed to $v$ in $\texttt{tree}_j$; and

(ii) since all nodes $\tau ay$ with $y\in B'$ are publicly committed to $1-v$, at least $\#B'\geq 2(t+1-|\tau a|)+1$ such nodes $\tau ay$ are committed to $1-v$ in $\texttt{tree}_j$ by the end of round $|\tau ax|+1$. It follows that $j$ will detect $x$ as faulty in round $|\tau|+3$.

If $x\in\mathcal{F}_j(|\tau|+3)$ then we are done. Otherwise, $x$ sent $j$ a $\texttt{mask}(x,a)$ message by the end of round $|\tau|+3$, and hence by the Masking rule we have that $\texttt{tree}_j(\tau bax)=1-v$,

so that $\tau bax$ is committed to $1-v$ in $\texttt{tree}_j$ by C1. In addition, since

(i) $\tau bax$ is not closed in $\texttt{tree}_j$ by the end of round $|\tau bax|+1$; and

(ii) $\tau bax$ is not fixed to $1-v$ by the end of round $|\tau bax|+2 = |\tau|+5$,

we have that $\tau bax$ is committed in $\texttt{tree}_j$ to $v$ by C3 by the end of round $|\tau|+5$. Given that $\tau ba$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$, Corollary 6.16 implies that $\tau ba$ is not closed in $\texttt{tree}_j$ by the end of round $|\tau|+5$. It follows that $\tau bax$ is not closed in $\texttt{tree}_j$ by the end of round $|\tau bax|+2 = |\tau|+5$. Hence, by fault discovery rule FD2, we obtain that $x \in \mathcal{F}_j(|\tau|+5)$ and we are done.
□

Roughly speaking, Lemma 7.3 implies that in a successful cross corruption, most members of $A$ must mask $a$ when reporting on $\tau ba$. The ones that do not will become disabled by the end of round $|\tau|+5$. As a result, if $F(|\tau|) \geq F(|\tau|+1)+3$, then for $\tau ba$ not to close by the end of round $|\tau|+7$, at least three members of $A$ must become disabled by the end of round $|\tau|+5$. As we shall see, this will be sufficient to guarantee that a cross corruption in these rounds must increase the waste by at least 1.

**7.1. A concrete admissible function.** Lemmas 7.2 and 7.3 motivate us to seek an admissible function $F$ with the following two properties:

1. $F(r) \leq t+r-4$ for all $2 \leq r \leq t$, and
2. $F(r) \geq F(r+1)+3$ for all $2 \leq r \leq t$.

In the notation of the above lemmas, the first property guarantees, for nodes $\tau$ satisfying $|\tau| \geq 2$, that if $\#B' \leq 2(t+1-|\tau a|)$ then $\#C \geq 3$. As we shall see, a combination of both properties implies that a cross corruption must necessarily cause an increase in the waste. Unfortunately, an admissible function with both properties does not exist. We now turn to define a function that will have the first property, and will approximate the second property: This property will hold for all but a small number of rounds $r$.

Recall that, for $F$ to be admissible, it must satisfy $t+r-1 \geq F(r) \geq t-r+2$ for $2 \leq r \leq t$. Intuitively, we divide the execution into "phases" consisting of many rounds each. In each phase we start with $F(r)$ being close to $t+r$, and we reduce the threshold by steps of 3 from one round to the next until we come close to $t-r$. A new phase then begins. Rather than at $t+r-1$, a phase will start with the threshold no greater that $t+r-4$, to guarantee the first property described above. Thus, both desired properties are maintained during a phase, but they are violated in the transition between phases. Luckily, the number of such transitions will be shown to be logarithmic in $n$.

We define $\mathsf{rem}(k)$ to be the difference between $k$ and the largest power of 2 that is smaller than $k$. More precisely stated,

$$\mathsf{rem}(k) \stackrel{\text{def}}{=} k - 2^{\lfloor \log_2 k \rfloor}.$$

Notice that $\mathsf{rem}(k)=0$ for $k \geq 1$ precisely if $k$ is a power of 2. Moreover, one property of $\mathsf{rem}$ that we shall use in the sequel is that, for all natural numbers $k \geq 1$ we have $0 \leq \mathsf{rem}(k) \leq \frac{k+1}{2}-1$, so that, in particular, we have $0 \leq 4\mathsf{rem}(k) \leq 2k-2$.

Our threshold function $F^*$ is defined as follows:

$$F^*(r) = \begin{cases} t+1 & \text{for } 0 \leq r \leq 4; \text{ and} \\ t+r-4-4\mathsf{rem}(r-3) & \text{for } r \geq 5. \end{cases}$$

A few essential properties of the function $F^*$ that we shall find useful in the sequel are:

LEMMA 7.4. *If $r \geq 5$ then*

$$t + r - 4 \geq F^*(r) \geq t - r + 4.$$

*Proof.* Since $0 \leq 4\mathsf{rem}(k) \leq 2k-2$ for $k \geq 2$, we have that $0 \leq 4\mathsf{rem}(r-3) \leq 2r-8$ for $r \geq 5$. It follows that $t+r-4 \geq F^*(r) \geq t+r-4-(2r-8) = t+r-4-2r+8 = t-r+4$ for $t+1 \geq r \geq 5$ and we are done. □

LEMMA 7.5. *The function $F^*$ is an admissible resolve function.*

*Proof.* Recall that a function $F$ is admissible if it satisfies $F(0) = F(1) = t+1$ and $t+r-1 \geq F(r) \geq t-r+2$ for $r \geq 2$. For $r = 0, 1$ we have $F^*(r) = t+1$ as desired. For $2 \leq r \leq 4$ we have $F^*(r) = t+1$ and $t+r-1 \geq t+1 \geq t-r+2$. For $r \geq 5$, Lemma 7.4 states that $t+r-4 \geq F^*(r) \geq t-r+4$. We thus have $t+r-1 \geq t+r-4 \geq F^*(r) \geq t-r+4 \geq t-r+2$ and we are done. □

LEMMA 7.6. *Let $r \geq 5$ and assume that $r-2$ is not a power of 2. Then $F^*(r) - F^*(r+1) = 3$.*

*Proof.* Since $r \geq 5$, we have that $F^*(r) = t+r-4-4\mathsf{rem}(r-3)$, while $F^*(r+1) = t+r-3-4\mathsf{rem}(r-2)$. The fact that $r-2$ is not a power of 2 implies that $\mathsf{rem}(r-2) = \mathsf{rem}(r-3)+1$. It follows that $F^*(r) = t+r-3+1-4(\mathsf{rem}(r-2)-1) = F^*(r+1) - 1 + 4 = F^*(r+1) + 3$ and we are done. □

Obviously, the number of rounds $r \leq t$ for which $r-2$ is a power of 2 is roughly $\log_2 t$.

**7.2. The main lemma.** We are now in a position to prove that, given our fixing and fault detection rules, the price of cross corruption is high. In other words, that for cross corruption to take place, many faulty processors must become disabled. As a result, cross corruption will no longer be a problem for monitor voting. We now have:

LEMMA 7.7. *Let $\tau a$ and $\tau b$ be universally corrupted nodes, such that*

*(i) $|\tau| \geq 4$ and $|\tau| - 1$ is not a power of 2;*

*(ii) the only universally corrupted child of $\tau a$ is $\tau ab$ and the only universally corrupted child of $\tau b$ is $\tau ba$; and*

*(iii) for some $i$, both of the nodes $\tau a$ and $\tau b$ are not closed in $\mathtt{tree}_i$ by the end of round $|\tau| + 7$.*

*Then*

$$\#(\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2)) \;\geq\; 5.$$

*Proof.* The situation considered in this lemma is partly illustrated by Figure 4. We first prove the following about $a$ and $b$.

CLAIM 7.8. $a, b \in (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$.

*Proof.* The fact that $\tau a$ and $\tau b$ are not closed in $\mathtt{tree}_i$ by the end of round $|\tau|+7$ implies, by Corollary 6.17, that these nodes are universally corrupted. By Corollary 6.19 we thus obtain that both $a$ and $b$ must be disabled no later than at the end of round $|\tau a| + 2 = |\tau| + 3$. However, if either of them were disabled by the end of round $|\tau| + 2$, then all nonfaulty processors would be masking them in round $|\tau| + 3$, and by Lemma 6.2 we would have that $\tau ba$ and $\tau ab$ would not be corrupted in any

nonfaulty processor's tree, contradicting our assumption about $a$ and $b$. It follows that

$$a, b \in (\mathcal{D}(|\tau|+3) \setminus \mathcal{D}(|\tau|+2)) \subseteq (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2)).$$

□

Given Claim 7.8, we need to show that, under the conditions of the lemma, at least 3 additional faulty processors must become disabled between rounds $|\tau|+3$ and $|\tau|+5$.

We shall perform our analysis based on the subtree rooted at $\tau a$. Assume the conditions of the lemma hold, and without loss of generality let $\mathtt{par}(|\tau a|) = v$. Since $\tau ab$ is the only universally corrupted child of $\tau a$, it follows by Corollary 6.17 that all other children of $\tau a$ must be fixed in $\mathtt{tree}_i$ by the end of round $|\tau|+6$. Let $A$ denote the set of processors $x$ such that the node $\tau ax$ is fixed to $v$ in $\mathtt{tree}_i$ (by the end of round $|\tau|+6$), and let $B$ the set of processors $y$ such that the node $\tau ay$ is fixed to $1-v$. This is depicted in Figure 4. The conditions of Lemma 7.1 are satisfied, and as a result we have that $\#A = F^*(|\tau a|) - 1$ and $\#B = n - |\tau a| - F^*(|\tau a|)$.

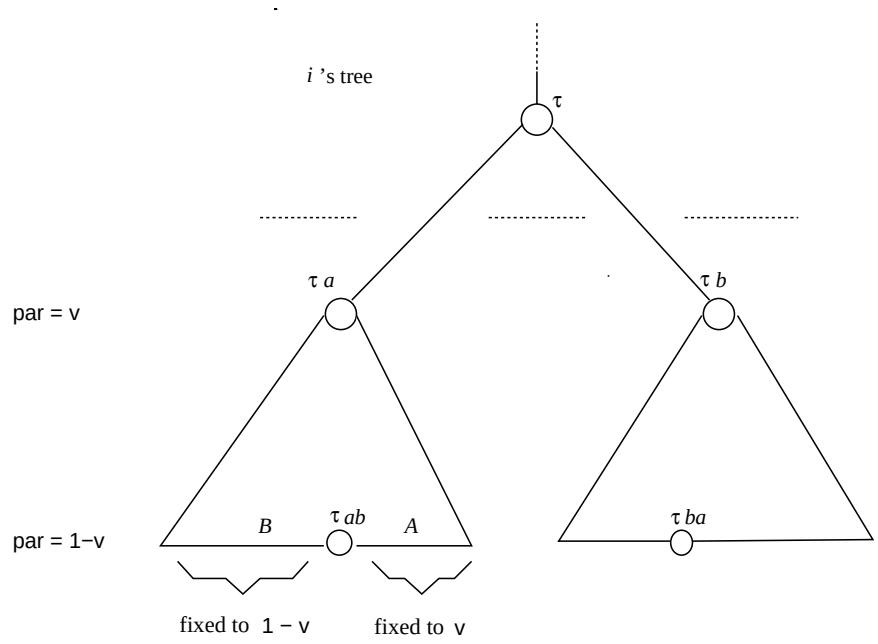


FIG. 4. *Two universally corrupted nodes.*

As in Lemma 7.2, let $C$ denote the subset of $B$ consisting of processors $c$ such that $\tau ac$ is publicly committed to $v$.

Let us denote $B' \stackrel{\text{def}}{=} B \setminus C$. By definition, every node $\tau ay'$ with $y' \in B'$ is publicly committed to $1-v$, and is hence committed to $1-v$ in every nonfaulty processor's tree. We now show that, if $B'$ is "small" (namely, $\#B' \leq 2(t+1-|\tau a|)$), then $C$ is large enough (i.e., $\#C \geq 3$) to yield the lemma. If $B'$ is not small, however, we shall use Lemma 7.3 to show that sufficiently many processors, this time members of $A$, must be disabled. Thus, either way we obtain that at least three processors in addition to $a$ and $b$ become disabled in rounds $|\tau|+3$ through $|\tau|+5$.

As described above, we consider two cases:

(i) $\#B' \leq 2(t+1-|\tau a|)$: In this case, we claim that $\#C \geq 3$. This follows from the fact that $\#B = n - |\tau a| - F^*(|\tau a|)$ and $\#C = \#B - \#B'$. The calculation is as follows.

$$\begin{aligned} \#C \geq n - |\tau a| - F^*(|\tau a|) - 2(t+1-|\tau a|) = \\ n - 2t + |\tau a| - 2 - F^*(|\tau a|) \geq \\ n - 2t + |\tau a| - 2 - (t + |\tau a| - 4) = \\ n - (3t+1) + 3 \geq 3 \end{aligned}$$

Lemma 7.2 implies that $C \subseteq (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$. In the claim above we showed that $a, b \in (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$. Since $a \notin C$ and $b \notin C$, we obtain that $\#(\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2) \geq 5$.

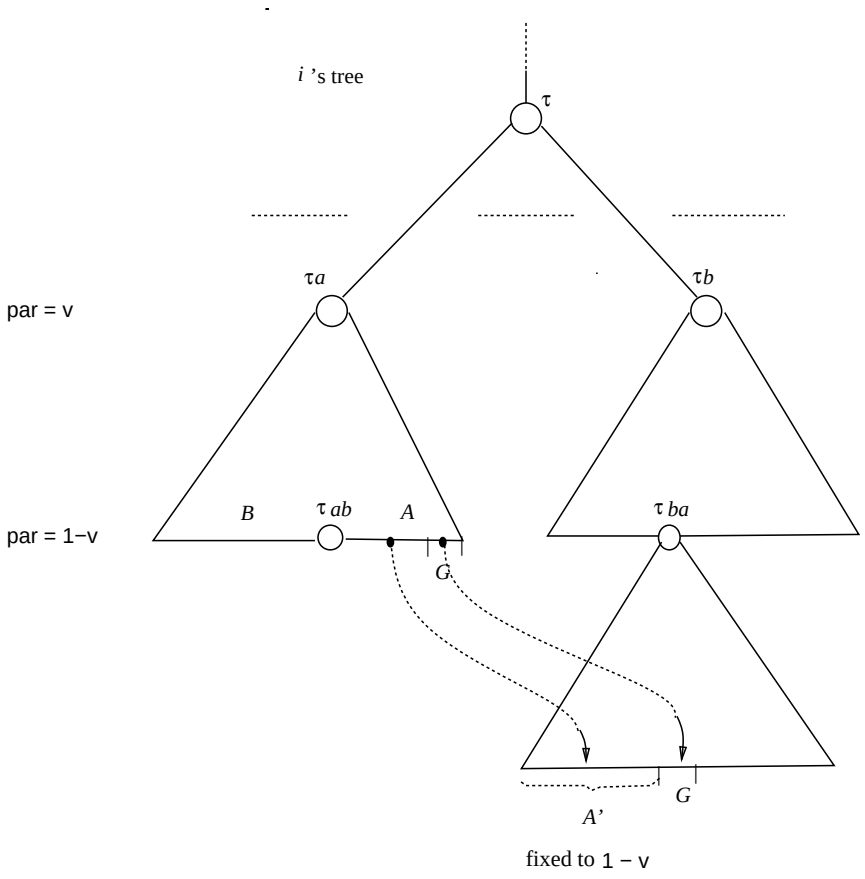


FIG. 5. *A set $G$ of newly disabled processors.*

(ii) $\#B' \geq 2(t+1-|\tau a|)+1$: Since we are assuming that $|\tau| \geq 4$ we have $|\tau a| \geq 5$. By Lemma 7.6 we thus have that $F^*(|\tau a|) - F^*(|\tau ba|) = 3$. Let $A'$ consist of the processors $x'$ such that $\tau bax'$ is fixed to $1-v$ in $\texttt{tree}_i$ by the end of round $|\tau|+7$. Since $\tau b$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$ and $\tau ba$ is the only universally corrupted child of $\tau b$, we have that $\tau ba$ is also not closed in $\texttt{tree}_i$ at that point. It follows that $\#A' \leq F^*(|\tau ba|) - 1 = F^*(|\tau a|) - 4 = \#A - 3$. Define $G \stackrel{\text{def}}{=} (A \setminus A')$ (see

Figure 5). In particular, we obtain that $\#G \geq 3$. Moreover, if a processor $z \in A$ is disabled by the end of round $|\tau| + 3$, then Lemma 6.2 and the Masking rules imply that $\tau baz$ is fixed in $\texttt{tree}_i$ to $\texttt{par}(|\tau ba|) = 1 - v$ by the end of round $|\tau| + 4$. It follows that no processor $x \in G$ is disabled by the end of round $|\tau| + 3$. The conditions of Lemma 7.3 are satisfied with respect to every $x \in G$, and hence by Lemma 7.3 we have that $G \subseteq (\mathcal{D}(|\tau| + 5) \setminus \mathcal{D}(|\tau| + 2))$. Notice that $a, b \notin G$. Thus, we again obtain that $\#(\mathcal{D}(|\tau| + 5) \setminus \mathcal{D}(|\tau| + 2)) \geq 5$, and we are done.
□

As a consequence of Lemma 7.7 and the definition of `Waste` we obtain:

COROLLARY 7.9. *If the conditions of Lemma 7.7 hold with respect to $\tau$, then* $\texttt{Waste}(|\tau| + 5) > \texttt{Waste}(|\tau| + 1)$.

*Proof.* We have defined $\texttt{Waste}(r)$ by:

$$\texttt{Waste}(r) \;=\; \#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r).$$

By definition, we have that $\texttt{correction}(r) \geq 0$, and $\texttt{deficit}(r) - \texttt{correction}(r) \geq 0$. It thus follows that

$$\#\mathcal{D}(r) + \texttt{deficit}(r) \geq \texttt{Waste}(r) + r \geq \#\mathcal{D}(r). \tag{1}$$

In particular, we have that $\texttt{Waste}(|\tau| + 5) + |\tau| + 5 \geq \#\mathcal{D}(|\tau| + 5)$. Recall that the deficit at $|\tau| + 1$ corresponds to processors that have universally corrupted nodes at depth $|\tau|$ but are not yet disabled at time $|\tau| + 1$. All of these processors are discovered as faulty by all nonfaulty processors, and are hence disabled, by time $|\tau| + 2$. Thus, we have that $\#\mathcal{D}(|\tau| + 2) \geq \#\mathcal{D}(|\tau| + 1) + \texttt{deficit}(|\tau| + 1)$. By equation (1) above we thus obtain that $\#\mathcal{D}(|\tau| + 2) \geq \texttt{Waste}(|\tau| + 1) + |\tau| + 1$.

By Lemma 7.7 we have that $\#(\mathcal{D}(|\tau| + 5) \setminus \mathcal{D}(|\tau| + 2)) \geq 5$. Since the set $\mathcal{D}$ grows monotonically, $\mathcal{D}(|\tau| + 5) \supseteq \mathcal{D}(|\tau| + 2)$, and we can therefore conclude that $\#\mathcal{D}(|\tau| + 5) \geq \#\mathcal{D}(|\tau| + 2) + 5$. In summary, we have

$$\texttt{Waste}(|\tau|+5)+|\tau|+5 \geq \#\mathcal{D}(|\tau|+5) \geq \#\mathcal{D}(|\tau|+2)+5 \geq \texttt{Waste}(|\tau|+1)+|\tau|+1+5.$$

Deducting $|\tau| + 5$ from both sides, we obtain that $\texttt{Waste}(|\tau| + 5) \geq \texttt{Waste}(|\tau| + 1) + 1$, so that $\texttt{Waste}(|\tau| + 5) > \texttt{Waste}(|\tau| + 1)$ and we are done. □

Lemma 7.7 implies that from round 5 on, the only rounds in which cross corruption can take place without increasing the waste of the run are pairs of rounds $r, r + 1$ such that $r - 1$ is a power of 2. In particular, since $r \leq t$, this can happen no more than $\log_2 t + O(1)$ times. These rounds can increase the number of universally corrupted nodes at any given depth in the tree by a factor of at most $O(t)$.

**7.3. Early stopping in $\Delta$-agreement.** A crucial property of the $\Delta$-EIG protocol is captured by Corollary 6.16. It states that two rounds after a node $\sigma$ is closed in one nonfaulty processor's tree, it will be closed in all processors' trees. Obviously, once $\sigma$ is closed in $\texttt{tree}_i$, processor $i$ has no use for the descendants of $\sigma$. Nevertheless, it might still need to record values and perform fault detection, in order to continue reporting on nodes in order to allow $\sigma$ to close in the trees of other processors. What Corollary 6.16 implies, then, is that $i$ needs to relay values in the subtree rooted at $\sigma$ for at most 2 rounds after $\sigma$ is closed in $\texttt{tree}_i$. This suggests that we can modify the $\Delta$-EIG protocol to obtain an early-stopping protocol as follows.

– Rather than reporting on all internal nodes in the $\Delta$-EIG tree, processor $i$ will report on a node $\sigma$ in round $|\sigma| + 1$ only if $\sigma$ was not closed in $\texttt{tree}_i$ by the end of round $|\sigma| - 2$. (Recall that a node can be closed before any value is stored in it;

all that is needed for it to be closed is that one of its ancestors should be fixed to a value.) To implement this rule, all that is needed is for $i$ to handle and report only on the children and grandchildren of nodes that fix by rule Fx2.

– We modify the Halting condition for a processor $i$ to:

• **Halting′:** Processor $i$ continues to record information, perform fault detection, and report on values for two rounds after the root $\lambda$ is closed in $\texttt{tree}_i$. At the end of these two rounds (and no later than at the end of round $t+1-\Delta$), it halts.

– There are a couple of details we need to take care of once we modify the protocol as described above. They result from the fact that it is now possible not to receive a message from a *nonfaulty* processor. This can only happen, however, in cases in which these values are of no use to the receiver. This leads to modifications of the Recording and masking rule, and to a modification of the definition of an ill-formatted message, which in turn affects the process of fault detection. We start by describing the latter. We shall extend the notion of an ill-formatted message as follows: If node $\sigma$ is not closed in $\texttt{tree}_i$ by the end of round $|\sigma|$, and processor $j$'s message in round $|\sigma|+1$ does not report values for all children of $\sigma$, then the message is considered ill-formatted and $i$ will detect $j$ as faulty by rule FD0. One consequence of this definition is that if $j$ sends no message to $i$ in round $r+1$ and the root $\lambda$ is not closed in $\texttt{tree}_i$ by the end of round $r$, then $i$ detects $j$ as being faulty.
Finally, the Recording and masking rule is modified for the case in which the root $\lambda$ is fixed in $\texttt{tree}_i$ at the end of round $r$ and $i$ receives no message in round $r+1$ from some processor $j \notin \mathcal{F}_i(r)$. Since $i$ needs to continue to report on values in round $r+2$ by the Halting′ rule, it acts as follows. For nodes that correspond to processors $z$ for which $j$ has issued $\texttt{mask}(j,z)$ reports in the first $r$ rounds, there is no problem. For all other nodes, we choose to have $i$ consider $j$ as reporting the same values that $i$ has reported, for all depth $r$ nodes that $i$ reports on in round $r+1$. This is related to the reconstruction method advanced by Zamsky [26, 28], and by Berman, Garay and Perry [7]. One feature of this choice is that it keeps the fault detection rule FD2 from ever causing $i$ to mistakenly "detect" $j$ as faulty. (While such a mistaken detection would not change $i$'s decision in the agreement process being executed, it becomes problematic when we run a number of agreement processes in parallel, and use a common fault-detection module as will be described in Section 8.1 below.)

We call the resulting protocol the $\Delta$-Es protocol (the Es stands for *early stopping*). Despite possibly reporting on much fewer nodes in a run of $\Delta$-Es than in similar runs of $\Delta$-Eig, the processors' behavior in $\Delta$-Es maintains an important invariant: If a node $\sigma$ is not closed in $\texttt{tree}_i$ at the end of round $|\sigma|$ and none of $\sigma$'s ancestors is closed in $\texttt{tree}_i$ by the end of round $|\sigma|+1$, then all nonfaulty processors report a value for $\sigma$ in round $|\sigma|+1$, as well as performing fault detection for all of $\sigma$'s children in round $|\sigma|+1$, and reporting on them in round $|\sigma|+2$. This ensures the following:

(i) For every nonfaulty $i$ and $j$, the first node to close in $\texttt{tree}_j$ along any path from the root, is guaranteed to close in $\texttt{tree}_i$ at most two rounds after it does in $\texttt{tree}_j$. Hence, the information that processor $i$ would receive in $\Delta$-Eig but does not receive in $\Delta$-Es does not affect $i$'s decision.

(ii) The commitment rules operate in $\Delta$-Es as they do in $\Delta$-Eig, since they depend only on the values stored in a node (C1), its children (C2) and grandchildren (C3). The fault detection rule FD2 remains sound, since it depends solely on these commitment rules.

(iii) The soundness of fault detection rule FD3 is maintained: If the node $\tau az$ is not closed in $\texttt{tree}_i$ by the end of round $|\tau az|+1$, then all processors are guaranteed

to store the children of $\tau a$ in the previous round, perform failure detection at the end of round $|\tau az|$, and report on nodes in round $|\tau az|+1$. Hence, a processor that does not mask the culprit processor $a$ of FD3 in the designated round, must be faulty.

As a result, all of the properties proven for $\Delta$-EIG in the previous sections hold once we move to the $\Delta$-ES protocol:

PROPOSITION 7.10. *All of the statements from Lemma 6.7 through Lemma 7.7 hold for the* $\Delta$-ES *protocol.*

The main aim of the protocol $\Delta$-ES is to allow us to refrain from having to construct exponential-size EIG trees and hence from having to send an exponential amount of communication. At this point, we are able to prove a polynomial relation between the number of universally corrupted nodes and the size of trees. This will reduce our problem to keeping the number of universally corrupted nodes polynomial, which will be done in the later sections. We thus have:

COROLLARY 7.11. *Assume that the last nonfaulty processor halts by the end of round* $r$*, and let the number of universally corrupted nodes* $\sigma$ *of depth* $|\sigma| \leq r-6$ *be* $T$*. Then, for every nonfaulty processor* $i$*, the total number of nodes in* $\texttt{tree}_i$ *is bounded by* $O(n^6 T)$*.*

*Proof.* Let $\sigma$ be a universally corrupted node. The node $\sigma$ has $O(n)$ children $\sigma j$ that are not universally corrupted. Corollary 6.17 implies that each such child is closed in $\texttt{tree}_i$ by the end of round $|\sigma j|+4$. It follows that $\sigma j$ has at most $O(n^4)$ descendants in $\texttt{tree}_i$ by the time it is closed. By definition of the $\Delta$-ES protocol, processor $i$ will need to store one level of nodes, beyond these $O(n^4)$ nodes, in the subtree rooted in $\sigma j$. It follows that $\sigma$ can have at most $O(n^6)$ descendants in $\texttt{tree}_i$ that are not themselves descendants of a universally corrupted child of $\sigma$. By accounting each node of $\texttt{tree}_i$ to its closest ancestor that is universally corrupted, the claim follows. □

**8. The `Sliding-flip` protocol.** We now have all of the ingredients necessary to define the final combined protocol. Intuitively, the protocol will be an variant of monitor agreement in which the monitors will be instances of $\Delta$-agreement. We remark that in such a setting, there is a distinction between the *global* round number, which is counted from the start of the original agreement process, and the *local* round number of a specific instance of $\Delta$-ES, which is counted from the start of this instance. The *global depth* and *local depth* of a node arefined analogously. The definition of $\Delta$-ES and our analysis of $\Delta$-ES use local round numbers. In our protocol, one instance of $\Delta$-agreement is spawned in every round, for rounds $1 \leq r \leq t$. In round $r$, the agreement process that is generated will be a $\Delta$-ES for $\Delta = r-1$. Since, by the Halting′ property, such an instance will complete in $t+1-\Delta = t+2-r$ rounds, we obtain that every such process will complete by the end of (global) round $r-1+t+2-r = t+1$. We call such a process *$(t+1)$-bounded*. Before we provide the details of the voting rules and properties of these instances of $\Delta$-agreement, we now describe a general method by which such agreement processes can be combined. This will be used when we come to combine the agreement processes in the final description of the combined protocol.

**8.1. Preempt-on-One.** In our combined protocol, we initiate one $\Delta$-agreement process $\texttt{M}^r$ in every round $r$, where the parameter $\Delta = r-1$ is used in $\texttt{M}^r$. In this section we describe the method by which we combine the different agreement processes. This method is stated in slightly more general terms than we need, as it applies elsewhere as well. We call a set of agreement processes being executed concurrently with respect to a single fault detection module (FDM) an *ensemble*, and denote ensembles by $\mathcal{E}$. Different agreement processes are said to use a single FDM if

for all nonfaulty processor $i$ and (global) round $r$, the set $\mathcal{F}_i(r)$ used by one agreement process is the same as that used by the others.

We now define an operation that takes an ensemble and turns it into a single protocol. In executing Preempt-on-One($\mathcal{E}$), a nonfaulty protocol concurrently executes all of the processes in $\mathcal{E}$. It decides and halts according to the following rule:

**Dec0.** Processor $i$ decides 0 on Preempt-on-One($\mathcal{E}$) and halts once $i$ has halted with a decision of 0 on all processes in $\mathcal{E}$.

**Dec1.** Processor $i$ decides 1 on Preempt-on-One($\mathcal{E}$) and halts (preempting all agreement processes underway) once $i$ has halted with a decision of 1 on some process in $\mathcal{E}$.

The following property of Preempt-on-One($\cdot$) is fairly immediate, and will turn out useful:

LEMMA 8.1. *Let $\mathcal{E}$ be an ensemble of agreement processes. If all processes in $\mathcal{E}$ are $(t+1)$-bounded and satisfy the Agreement property, then* Preempt-on-One($\mathcal{E}$) *is $(t+1)$-bounded and satisfies the Agreement property.*

*Proof.* We start by showing that Preempt-on-One($\mathcal{E}$) is an agreement protocol. Assume that some processor $i$ decides 0 on Preempt-on-One($\mathcal{E}$) in a given run. It follows from Dec0 that $i$ decides 0 on all processes in $\mathcal{E}$ in this run. Since the instances all satisfy the Agreement property, no other nonfaulty processor $j$ will decide 1 on any of the processes in $\mathcal{E}$. In addition, since the processes in $\mathcal{E}$ are all $(t+1)$-bounded, it follows that $j$ will actually decide 0 on every process in $\mathcal{E}$ no later than in round $t+1$. We conclude that every nonfaulty processor will halt in Preempt-on-One($\mathcal{E}$) with a decision of 0 by the end of round $t+1$. Assume now that $i$ decides 1 on Preempt-on-One($\mathcal{E}$) in a given run. Let $\mathtt{M} \in \mathcal{E}$ be the process that triggers $i$'s decision. In particular, it follows that processor $i$ decides 1 and then halts on M at the end of some round $r_i \leq t+1$. Let $j \neq i$ be any other nonfaulty processor. There are two possibilities: (i) Processor $j$ decides and halts on Preempt-on-One($\mathcal{E}$) before it has a chance to decide and halt on the process M. By rule Dec0 in the definition of Preempt-on-One($\mathcal{E}$), a decision of 0 can only be taken after $j$ has decided and halted on all processes in $\mathcal{E}$. It follows that $j$ could only have decided 1, so its decision is in agreement with processor $i$'s decision. Moreover, since M was $(t+1)$-bounded, and $j$ decided before it had a chance to decide on M, we obtain that $j$ decides before round $t+1$. (ii) The other possibility is that $j$ does manage to decide and halt on M. Here again because M satisfies the Agreement property, $j$ will decide 1 on Preempt-on-One($\mathcal{E}$). In addition, since M was $(t+1)$-bounded, $j$ decides no later than in round $t+1$. In summary we have that in all cases, Preempt-on-One($\mathcal{E}$) satisfies the Agreement property and is $(t+1)$-bounded, and we are done. □

**8.2. Putting it all together.** As described above, the ensemble generated in the combined protocol consists of instances of $\Delta$-ES protocols, initiated one per round, in rounds $1 \leq R \leq t$. We use $R$ in this section to refer to global round numbers. In round $R$, a monitor process initiated in global round $K$ will have its own local round number $r = R + 1 - K$. All the instances of $\Delta$-ES protocols invoked use the function $F^*$ defined in Section 7.1. Notice that the function $F^*$ is applied, for every instance of $\Delta$-ES, relative to the local round count. Thus, in the same global round, each agreement process has its own local round number. The combined protocol will be Preempt-on-One($\mathcal{E}$), for the ensemble thus generated. We now describe how to determine a processor's initial vote in any given monitor process.

Clearly, in round $R = 1$, the value a processor $i$ uses as its initial value is its original initial value $v_i$. In later rounds $R > 1$, the initial value of processor $i$ in

monitor process $\mathtt{M}^R$, is essentially the same as the one we mentioned in Section 4.3 for $n > 4t$. For the purpose of this voting rule, we say that processor $i$ has *detected* another processor $z$ *as being disabled* by the end of round $K$ if $i$ has received $\mathtt{mask}(j, z)$ reports from at least $2t + 1$ processors $j$ by that point. The following lemma justifies this terminology.

LEMMA 8.2. *If $i$ has detected $z$ as being disabled by the end of round $K$, then $z$ is disabled at the end of round $K$.*

*Proof.* By definition, $i$ can detect $z$ as being disabled only once $i$ has received at least $2t+1$ $\mathtt{mask}(j, z)$ reports. At least $t+1$ of these reports are from nonfaulty processors $j$. The sending rule implies that a nonfaulty processor sends identical messages to all processors in every round. It thus follows that every nonfaulty processor $i'$ must have received at least $t + 1$ $\mathtt{mask}(j, z)$ reports by the end of round $K$. As a result, the fault detection rule FD1 implies that each such processor $i'$ must detect $z$ as being faulty by the end of round $K$. It follows that $z$ is disabled at that point, and we are done. □

The voting rule on a monitor $\mathtt{M}^R$ with $R > 1$ is:

**Monitor-vote:** Processor $i$ will vote 1 on $\mathtt{M}^R$ if

(i) at least one node at (global) depth $R - 1$ in one of $i$'s trees is not closed; and

(ii) $i$ has detected at least $R - 1$ faulty processors as being disabled by the end of round $R - 1$.

It will vote 0 on $\mathtt{M}^R$ otherwise.

We define the `Sliding-flip` protocol to be the protocol that results from performing `Preempt-on-One` on the ensemble consisting of the original agreement tree together with the monitor agreement processes $\mathtt{M}^R$, for $2 \leq R \leq t$, where the votes of the nonfaulty processes are obtained according to the Monitor-vote rule above. These agreements processes are all instances of $\Delta$-ES, based on the threshold function $F^*$ defined in Section 7.1. For ease of exposition, we shall consider the original agreement process to be $\mathtt{M}^1$.

The role of part (i) in the Monitor-vote rule is to guarantee that if all initial values are 0, then a decision of 0 will be reached. The role of part (ii) is to guarantee that the monitors satisfy the initial conditions of $\Delta$-agreement: A nonfaulty process votes 1 on a monitor agreement process only if a sufficient number of processors are disabled. In fact, an immediate consequence of Lemma 8.2 is:

COROLLARY 8.3. *If at least one nonfaulty processor votes* 1 *in a monitor* $\mathtt{M}^{R+1}$, *then* $\#\mathcal{D}(R) \geq R$.

Corollary 8.3 formally shows that the monitors that are initiated in the combined protocol are "legal" instances of $\Delta$-agreement. We can now prove:

PROPOSITION 8.4. *The* `Sliding-flip` *protocol is a correct* $(t + 1)$*-round Byzantine agreement protocol.*

*Proof.* Theorem 6.10 and Corollary 8.3 imply that all of the agreement processes initiated in `Sliding-flip` are $(t+1)$-bounded instances of Byzantine agreement. We therefore obtain by Lemma 8.1 that the `Sliding-flip` protocol satisfies the Decision and Agreement conditions. We need to show that it also satisfies Validity. If all initial values are 1, then the root of the initial tree will fix to 1 at the end of round 2 for all of the nonfaulty processors, so by the definition of `Preempt-on-One` they will all decide 1. Assume that all initial values are 0. In particular, the initial values of all nonfaulty processors are 0. It follows that, for every nonfaulty processor $i$, at least $2t+1$ of the root's children in the initial tree will store 0 at the end of round 1. Since 0

is the preferred value in this round, the root will fix to 0. As a result, all nodes at global depth 1 will be closed for every nonfaulty processor, and by the Monitor-vote rule, every such processor $i$ will vote 0 on $\mathtt{M}^2$. A straightforward induction on $R$ now shows that all nonfaulty processors vote 0 on all monitors $\mathtt{M}^R$, so that all monitors decide 0, and by definition of `Preempt-on-One` every nonfaulty processor $i$ will end up deciding 0 at time $t+1$. □

It remains to show that the protocol is efficient. As a partial converse of Lemma 8.2 we have

LEMMA 8.5. *If $z$ is disabled at the end of round $R$ and no nonfaulty processor has halted by the end of round $R$, then every nonfaulty processor $i$ will have detected $z$ as being disabled by the end of round $R+1$.*

*Proof.* If no nonfaulty processor has halted by the end of round $R$, then they all send messages in round $R+1$. If $z$ is disabled at the end of round $R$, then every nonfaulty processor $j$ must send a $\mathtt{mask}(j,z)$ message to $i$ no later than in round $R+1$. It follows that $i$ is guaranteed to receive at least $2t+1$ such messages by the end of round $R+1$, and we are done. □

Recall that the `Waste` was defined with respect to a given instance of $\Delta$-ES. The parameter $r$ of $\mathtt{Waste}(r)$ in a given instance of $\Delta$-ES is a local round number. When we run many instances of $\Delta$-ES concurrently, as in the `Sliding-flip` protocol, we wish to reason about an overall notion of waste. Let us define the global waste at time $K$, denoted by $\mathtt{G\_Waste}(K)$, to be the maximum value of $\mathtt{Waste}(K-R)$ in monitor $\mathtt{M}^R$, over all $R < K$. Thus, $\mathtt{G\_Waste}(K)$ is the maximal value that the waste obtains at global time $K$. As a consequence of Lemma 8.5 and the monitor voting rule, we now have:

LEMMA 8.6. *If $\mathtt{G\_Waste}(R) \geq 2$ and some node is universally corrupted, then all nonfaulty processors are guaranteed to halt by the end of round $R+6$.*

*Proof.* If $R+6 \geq t+1$ the claim is immediate from the definition of the combined protocol, which is guaranteed to halt for every processor by the end of (global) round $t+1$. Assume $R+6 < t+1$. It follows that processors can halt by the end of round $\leq R+6$ only due to their deciding 1 on some monitor. First assume that some nonfaulty processor $i$ halts by the end of round $R+4$, based on deciding 1 on a monitor $\mathtt{M}^{R'}$ for some $R' < R+3$ by the end of round $R+2$. It follows from Corollary 6.16 and Proposition 7.10 that all processors will have decided 1 on $\mathtt{M}^{R'}$ by the end of round $R+4$ and will halt by the end of round $R+6$. Assume now that no nonfaulty processor has halted by the end of round $R+4$ due to a monitor $\mathtt{M}^{R'}$ with $R' < R+3$. The fact that $\mathtt{G\_Waste}(R) \geq 2$ implies that $\#\mathcal{D}(R+1) \geq R+2$. Lemma 8.5 states that every nonfaulty processor $i$ detects all members of $\mathcal{D}(R+1)$ as being disabled by the end of round $R+2$. As a result, $i$ will have detected at least $R+2$ disabled processors by the end of round $R+2$. Finally, the fact that some node is universally corrupted implies that, for every nonfaulty processor $i$, one of $i$'s trees has survived the first round. The monitor voting rule now states that $i$ will vote 1 on $\mathtt{M}^{R+3}$. Since all of the processors vote 1 on $\mathtt{M}^{R+3}$ and are active until the end of round $R+4$, it follows that they all decide 1 on $\mathtt{M}^{R+3}$ at the end of round $R+4$ and will thus halt by the end of round $R+6$. □

Given Lemma 8.6 and Theorem 6.23, we can now prove:

LEMMA 8.7. *Choose $L$ such that the last nonfaulty processor halts by the end of round $L+6$. Then, for every nonfaulty $i$ and every monitor $\mathtt{M}^R$ with $R \leq L$, the number of universally corrupted nodes in $\mathtt{tree}_i$ for $\mathtt{M}^R$ is $O(t^2)$.*

*Proof.* If, for some nonfaulty processor $i$, none of $i$'s trees survives the first round,

then no node is universally corrupted, and we are done. We shall therefore assume from now on that at least one node is universally corrupted. By Lemma 8.6, the waste cannot reach 2 before round $L$, or all processors would halt before round $L+6$. We thus reason about the rounds preceding $L$, assuming the waste never reaches 2 in those rounds. We claim that if any node of (global) depth $K \geq 2$ is universally corrupted, then $\texttt{G\_Waste}(K) \geq -2$. Since, for every monitor it is guaranteed by definition that $\texttt{deficit}(r) - \texttt{correction}(r) \geq 0$ for every $r$, it suffices to show that $\#\mathcal{D}(K) - K \geq -1$. Assume that $\sigma$ is a node of depth $K$ in a monitor $\texttt{M}^R$ (and hence $|\sigma| = K-(R-1)$). Since there is a universally corrupted node in $\texttt{M}^R$, we know that at least one nonfaulty processor voted 1 on this monitor, and hence by the Monitor-vote rule and by Lemma 8.2 we have that $\#\mathcal{D}(R-1) - (R-1) \geq 0$. If $|\sigma| \leq 2$ we are clearly done. If, however, $|\sigma| \geq 3$, then the path from the root to $\sigma$ consists of nodes all of which are universally corrupted. Clearly, by the end of round $K$, all processors universally corrupting nodes at depths smaller than $K-1$ are disabled. It follows that $|\sigma|-2$ of $\sigma$'s ancestors contribute fresh disabled processors that were not in $\mathcal{D}(R-1)$. We thus obtain that $\mathcal{D}(K) \geq R-1+|\sigma|-2 = R-1+K-(R-1)-2 = K-2$, and hence $\mathcal{D}(K) - K \geq -2$ and the claim is proven. We conclude that $\texttt{G\_Waste}$ can vary within a small constant range without causing a monitor to halt the protocol.

It is straightforward to check that if $k \geq 3$ processors universally corrupt nodes at (global) depth $R$, then $\texttt{G\_Waste}(R+1) - \texttt{G\_Waste}(R-1) \geq k-2$. It follows that this can happen only a constant number of times. A round with no universal corruption whatsoever will cause all of the active monitors to close in two rounds. As argued in Section 7.2 following Corollary 7.9, the only case in which two processors can corrupt nodes at a given depth in the same tree without increasing the waste is when cross corruption takes place in a pair of rounds $r, r+1$ such that $r-1$ is a power of 2. The number of such rounds $r \leq t+1$ is $\log t + O(1)$. It follows that the number of universally corrupted nodes at any particular level of $\texttt{tree}_i$ for a monitor $\texttt{M}^R$ is $O(t)$. Hence, the total number of universally corrupted nodes created by the end of round $L$ in $\texttt{tree}_i$ for $\texttt{M}^R$ is bounded by $O(t^2)$. □

As a consequence of Lemma 8.7 and Lemma 7.11 we obtain

LEMMA 8.8. *For every nonfaulty processor $i$, the total size of each of the $t$* EIG *trees that $i$ ever constructs is polynomial.*

Finally, as a result of Lemma 8.8 and Corollary 7.11 we have:

THEOREM 8.9. *The* `Sliding-flip` *protocol is a correct Byzantine agreement protocol that halts in $t+1$ rounds in the worst case, and is polynomial in both communication and computation.*

**9. Early stopping.** The `Sliding-flip` protocol consists of an ensemble of agreement processes running concurrently. They are initiated one per round, and are combined using the `Preempt-on-One` scheme: a local decision of 1 in any instance causes a global decision of 1 coupled with preemption of all decision processes. A decision of 0 can be reached only at time $t+1$, in case all agreement processes turn out to have decided 0. (Recall that the agreement processes in the ensemble are all guaranteed to reach a local decision by the end of round $t+1$.) In particular, even in runs with no failures, a decision of 0 cannot take place before round $t+1$. Thus, while `Sliding-flip` is a polynomial protocol that is guaranteed to halt in $t+1$ rounds, it is not guaranteed to stop early in runs in which few processors actually fail. In this section we discuss how to modify the `Sliding-flip` protocol to obtain a protocol that does stop early when few failures actually occur. The concept of *early stopping* is due to [13], who showed that no protocol for Byzantine agreement can be guaranteed to

halt in fewer than $\min\{t+1, f+2\}$ rounds in the worst case, where $f$ is the number of failures that occur in the run in question. Our goal will be to obtain a protocol that is guaranteed to halt in $\min\{t+1, f+c\}$ rounds for a small constant $c$.

The `Preempt-on-One` scheme already takes care of stopping quickly when the decision is 1. Early decision on 1 was of crucial importance to the `Sliding-flip` protocol, because that was the way the protocol keeps trees from growing beyond a polynomial bound. What remains, therefore, is to allow early stopping on a decision of 0 without hindering the correctness of the early decision on 1. Our basic strategy will be to maintain the basic `Preempt-on-One` rules for deciding on 1. Early stopping on 0 will then depend on our ability to predict at an early stage that all agreement processes that have been initiated, as well as all those that are due to be initiated in the future, are bound to decide 0. It is safe to decide 0 when this happens, rather than wait until the end of round $t+1$ to do so. The following lemma describes a fairly general condition that guarantees that all future monitors will decide 0:

LEMMA 9.1. *In the protocol* `Sliding-flip`*, if all monitors* $\mathtt{M}^R$ *with* $R \leq K$ *are closed on* 0 *for all nonfaulty processors at the end of round* $K$*, then every monitor* $\mathtt{M}^R$ *with* $R > K$ *that is initiated in the protocol closes on* 0 *at the end of its first round* $R$*.*

*Proof.* Assume that all monitors $\mathtt{M}^R$ with $R \leq K$ are closed on 0 for all nonfaulty processors at the end of round $K$. Part (i) of the Monitor-vote rule implies that all nonfaulty processors will vote 0 on the monitor $\mathtt{M}^R$ for the first $R > K$. This monitor will close on 0 in its first round for all nonfaulty processors, and the same argument can now be applied inductively to show that all later monitors will do the same. □

The problem in trying to apply the condition of Lemma 9.1 is that this condition is not one that can be detected by an individual processor. We now discuss a way of making a similar condition detectable. In order to do so, we consider a variant `Sliding-flip`$'$ of the `Sliding-flip` protocol that differs from the original only in that, instead of initiating a new monitor agreement process $\mathtt{M}^R$ in every round, such a process is initiated once in 5 rounds. Specifically, $\mathtt{M}^R$ monitors are initiated for every integer $R$ of the form $R = 5k+1$, for $1 \leq k \leq \lfloor \frac{t-1}{5} \rfloor$. (The agreement tree based on the processors' original initial values is, of course, initiated in round 1.) First notice that `Sliding-flip`$'$ has all of the desired qualities of `Sliding-flip`: It halts in at most $t+1$ rounds, and is guaranteed to be polynomial. The proof of correctness of `Sliding-flip`$'$ is the same as that for `Sliding-flip`. The only changes required in order to prove the complexity bounds for `Sliding-flip`$'$ are in Corollary 7.11, Lemma 8.6 and Lemma 8.7. In Lemma 8.6 we are now guaranteed only that if $\mathtt{G\_Waste}(R) \geq 6$ then a monitor issued no later than round $R+7$ closes with value 1. Without any fine tuning, this will cause an increase of $O(n^4)$ in the complexity of the protocol. The analogue of Corollary 7.11 will replace $r-6$ by $r-10$ and $n^6T$ by $n^{10}T$. The proof of the analogue of Lemma 8.7 goes through essentially without change, when we assume $\mathtt{G\_Waste}(R) \leq 6$ instead of assuming that $\mathtt{G\_Waste}(R) \leq 2$.

Define $\Phi_i(K)$ to hold if (i) $K \equiv 1 \bmod 5$, (ii) no node of (global) depth $K$ is corrupted in any of $i$'s trees, and (iii) no agreement process initiated in a round $R \leq K$ either has or will close with the root fixed to value 1. Intuitively, we will use $\Phi_i(K)$ to determine a condition for early stopping on 0. We can show:

LEMMA 9.2. *If* $\Phi_i(K)$ *holds for some* $K$*, then all monitors* $\mathtt{M}^R$ *for* $R > K$ *that will ever be initiated in* `Sliding-flip`$'$ *will close with value* 0 *in round* $R$*, for all nonfaulty processors.*

*Proof.* Let $K \equiv 1 \bmod 5$, and assume that no node of (global) depth $K$ is corrupted in any of $i$'s trees. Notice that because $K \equiv 1 \bmod 5$, no monitors are initiated

in the rounds $K+1, \ldots, K+4$. By definition of corruption, the fact that no node of (global) depth $K$ is corrupted for $i$ implies that all depth $K$ nodes in all of $i$'s trees are closed by the end of round $K+2$. Corollary 6.16 implies that all depth $K$ nodes in the trees of all nonfaulty processors are closed by the end of round $K+4$. It follows that all trees corresponding to existing agreement processes close for all nonfaulty processors by that time. As a result, by part (i) of the Monitor-vote rule, all nonfaulty processors will vote 0 on $\mathtt{M}^R$ for $R = K+5$, and a straightforward inductive argument shows that they will all vote 0 on every monitor initiated thereafter. □

The condition $\Phi_i(K)$ used in Lemma 9.2 is easily detectable by process $i$. Moreover, Lemma 9.2 implies that if $\Phi_i$ holds, then $\Phi_j$ will hold soon thereafter:

COROLLARY 9.3. *Let $i$ and $j$ be nonfaulty processors. If $\Phi_i(K)$ holds in a run of* `Sliding-flip'`*, then $\Phi_j(K+5)$ holds at the end of round $K+5$.*

*Proof.* Assume that $\Phi_i(K)$ holds. The argument given in the proof of Lemma 9.2 shows that the trees corresponding to all monitors initiated in rounds $R \leq K$ will be closed for all nonfaulty processors by the end of round $K+4$. By the Monitor-vote rule, all nonfaulty processors vote 0 on the monitor initiated in round $K+5$. As a result, this monitor is closed at the end of round $K+5$, for all nonfaulty processors. It follows that $\Phi_j(K+5)$ holds, for all nonfaulty processors $j$. □

Notice that if $\Phi_i(K)$ holds, then $i$ will detect that $\Phi_i(K)$ holds no later than by the end of round $K+2$. Moreover, the proof of Corollary 9.3 shows that $\Phi_j(K+5)$ will be detectable by $j$ no later than by the end of round $K+5$. It follows that once $i$ detects that $\Phi_i(K)$ holds, it can essentially decide 0. The only thing it needs to do in order to guarantee that $\Phi_j$ will hold for all $j$ is to continue to participate in the existing agreement processes (all of which, by Corollary 6.17, are guaranteed to terminate by the end of round $K+4$), and send its vote in the monitor initiated in round $K+5$. This provides an early-stopping method for deciding 0.

We define the `ES-Sliding-flip` protocol to consist of `Sliding-flip'` with the following modifications: (i) we add the rule that once a processor $i$ first detects that $\Phi_i(K)$ holds, it decides 0 and it halts once all of its monitors initiated in rounds $R \leq K$ halt according to $\Delta$-ES; and (ii) we add the default that sending no value as a vote on a monitor amounts to sending a vote of 0 (and is not regarded an ill-formed message). The purpose of point (ii) is to enable a processor that detects that $\Phi_i(K)$ holds to halt before round $K+5$. Its silence in round $K+5$ will be interpreted as a vote of 0, which will suffice to ensure that the necessary conditions of $\Phi_j(K+5)$ will hold. We can now summarize the properties of the protocol `ES-Sliding-flip` as follows:

THEOREM 9.4. *The* `ES-Sliding-flip` *protocol is a correct Byzantine agreement protocol that is polynomial in both communication and computation. It halts in $\min\{t+1, f+5\}$ rounds in the worst case, where $f$ is the number of failures that actually take place.*

*Proof.* Correctness of the protocol `ES-Sliding-flip` is inherited from that of `Sliding-flip'` and Corollary 9.3, which guarantees that if one of the nonfaulty processors decides 0, then all of them will. The polynomial complexity and $(t+1)$-boundedness of `ES-Sliding-flip` are also inherited from `Sliding-flip'`. It remains to show that the protocol halts in no more than $f+5$ rounds. First notice that for a node of global depth $K$ to be corrupted, at least $K$ distinct processors must fail. This is obvious in the initial agreement process, and is true for arbitrary monitors $\mathtt{M}^R$ based on Lemma 6.1 and part (ii) of the Monitor-vote rule. Assume that exactly $f$ processors fail in a given run. It follows that no more than $f$ processors can ever be

disabled, and no node of depth greater than $f$ can be corrupted. It follows that by the end of round $f+3$ all monitors ever initiated are closed in all nonfaulty processors' trees. Since exactly $f$ processors fail, if a node of depth $f$ is corrupted, no nonfaulty processor can detect $f$ processors as being disabled before round $f+2$, because an additional round in which `mask` messages are sent is necessary for the detection. This implies that the last monitor on which some nonfaulty processor votes 1 can be initiated in a round $K \leq f$. Let $F = \min\{K : K \equiv 1 \bmod 5 \,\&\, K > f\}$. Notice that $F \leq f+5$. Let $i$ be an arbitrary nonfaulty processor. If $\Phi_i(K)$ holds for some $K < F$, then we have seen that $i$ detects $\Phi_i(K)$ no later than time $K+2$, and halts no later than time $K+4$, and $K+4 < F \leq f+5$. Assume that $\Phi_i(K)$ did not hold for $K < F$. It follows that $\Phi_i(F)$ will hold at the end of round $f+3$. Processor $i$ will be able to decide 0 at the end of round $f+3$, and halt at the end of round $f+5$. It follows that all processors decide and halt by the end of round $f+5$ and we are done. □

A few remarks are in order:

(i) It is possible to use the reconstruction method of [26, 28, 7] in order to avoid the need for the additional two rounds in which nonfaulty processors "echo" values to ensure that the others reach the same decision in an instance of $\Delta$-ES. A variant of `ES-Sliding-flip` based on such a modified version of $\Delta$-ES will halt in $\min\{t+1, f+3\}$ rounds.

(ii) To obtain an early-stopping protocol halting within $f+2$ rounds, what is needed is to extend the fixing rule `Fx2` in order to allow fixing in round $|\sigma|+1$ to the non-favored value $1 - \mathtt{par}(|\sigma|)$ if an overwhelming majority of $\sigma$'s children store this value. The current version allows fixing only to $\mathtt{par}(|\sigma|)$. The current choice was made in order to simplify the statements of the various lemmas in this paper, and to shorten their proofs. We believe that with this extension and using reconstruction, it should be possible to obtain an early stopping protocol that halts in the optimal bounds of $\min\{t+1, f+2\}$ rounds.

(iii) In runs with no failures, our protocol decides in at most three rounds and halts in five. Only a single agreement tree is ever constructed, and the protocol acts just like a simple straightforward protocol. With reconstruction the protocol would halt in three rounds, and with an extended `Fx2` rule both decision and termination would be obtained in two rounds.

**Acknowledgements.** The authors would like to thank Café Ka'ze in Tel-Aviv for providing the ambiance and hospitality during critical stages of this work. Special thanks to Cynthia Dwork, Arkady Zamsky and to an anonymous referee for comments and suggestions that improved this paper. The first author is also thankful to Piotr Berman for innumerable discussions on the subject, and the second author is similarly thankful to Orli Waarts.

## REFERENCES

[1] A. Bar-Noy and D. Dolev, *Consensus Algorithms with One-Bit Messages*, *Distributed Computing*, 4 (1991), pp. 105-110. Preliminary version appeared as *Families of Consensus Algorithms* in Proceedings of the 3rd Aegean Workshop on Computing, LNCS(319), pp. 380-390, Springer-Verlag, Berlin, 1988.

[2] A. Bar-Noy, D. Dolev, C. Dwork and H.R. Strong, *Shifting gears: changing algorithms on the fly to expedite Byzantine Agreement*, Information and Computation, 97:2 (1992) pp. 205–233.

[3] P. Berman and J.A. Garay, *Cloture votes: $n/4$-resilient distributed consensus in $t+1$ rounds*, *Mathematical Systems Theory*, special issue dedicated to fault-tolerant

distributed algorithms (ed. Ray Strong), 26 (1993), pp 3–20. A preliminary version appears as part of [6].

[4] ———*Efficient distributed consensus with $n = (3 + \epsilon)t$ processors*, in Proceedings of the 5th International Workshop on Distributed Algorithms, LNCS (579), pp. 129-141, Springer-Verlag, Berlin, October 1991.

[5] ———unpublished manuscript, 1990.

[6] P. Berman, J.A. Garay and K.J. Perry, *Towards optimal distributed consensus*, *Proc. 30th FOCS*, pp. 410–415, October/November 1989. A revised version of the *Cloture Votes* algorithm appears as [3].

[7] ———*Optimal early stopping in distributed consensus*, *Proc. 6th International Workshop on Distributed Algorithms*, LNCS (647), pp. 221–237, Springer-Verlag, Berlin, November 1992.

[8] B. Coan, *A communication-efficient canonical form for fault-tolerant distributed protocols*, in Proceedings of the 5th ACM Symposium on the Principles of Distributed Computing, 1986, pp. 63–72.

[9] ———*Efficient agreement using fault diagnosis*, *Distributed Computing*, 7(1993), pp. 87–98. Preliminary version appeared in Proceedings of the 26th Allerton Conference on Communication, Control and Computing, pp. 663-672, University of Illinois, Urbana, Ill., 1988.

[10] B. Coan and J. Welch, *Modular construction of an efficient 1-Bit Byzantine agreement protocol*, *Mathematical Systems Theory*, special issue dedicated to fault-tolerant distributed algorithms (ed. H.R. Strong), 26 (1993), pp. 131–154.

[11] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch and H.R. Strong, *An efficient algorithm for Byzantine agreement without authentication*, Information and Control, 52 (1982), pp. 257–274.

[12] C. Dwork and Y. Moses, *Knowledge and common knowledge in a Byzantine environment: crash failures*, Information and Computation, 88 (1990), pp. 156–186.

[13] D. Dolev, R. Reischuk and H.R. Strong, *Early stopping in Byzantine agreement*, IBM Research Report RJ5406 (55357), IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, 1986. Revised version appears in J. Assoc. Comput. Mach., 37 (1990), pp. 720–741.

[14] D. Dolev and H.R. Strong, *Polynomial algorithms for multiple processor agreement*, in Proceedings of the 14th Annual ACM Symposium on Theory of Computing, pp. 401–407, ACM Press, New York, NY, 1982.

[15] M. J. Fischer, *The consensus problem in unreliable distributed systems (a brief survey)*, Yale University Technical Report YALEU/DCS/RR-273, 1983.

[16] M. J. Fischer and N. A. Lynch, *A lower bound for the time to assure interactive consistency*, Inf. Proc. Letters, 14 (1982), pp. 183–186.

[17] P. Feldman and S. Micali, *Optimal algorithms for Byzantine agreement*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 148–161, ACM Press, New York, NY, 1988.

[18] J. A. Garay and Y. Moses, *Fully polynomial Byzantine agreement in $t+1$ rounds*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, pp. 31–41, ACM Press, New York, NY, 1993.

[19] O. Goldreich and E. Petrank, *The best of both worlds: guaranteeing termination in fast randomized Byzantine agreement protocols*, Inf. Proc. Letters, 36 (1990), pp. 45–49.

[20] L. Lamport, R.E. Shostak and M. Pease, *The Byzantine generals problem*, ACM Trans. Prog. Lang. and Systems, 4:3 (1982), pp. 382–401.

[21] Y. Moses and O. Waarts, *Coordinated traversal: $(t+1)$-round Byzantine agreement in polynomial time*, J. of Algorithms, 17:2 (1994), pp. 110–156. An extended abstract appeared in Proceedings of the 29th IEEE Symposium on the Foundations of Computer Science, pp. 246–255, 1988.

[22] M. Pease, R. Shostak and L. Lamport, *Reaching agreement in the presence of faults*, J. Assoc. Comput. Mach., 27 (1980), pp. 121–169.

[23] M. Rabin *Randomized Byzantine generals*, in Proceedings of the 24th IEEE Symposium on the Foundations of Computer Science, pp. 403-409, IEEE Press, 1983.

[24] S. Toueg, K.J. Perry and T.K. Srikanth, *Fast distributed agreement*, SIAM J. Comput., 16, (1987) pp. 445–457.

[25] O. Waarts, *Coordinated traversal: Byzantine agreement in polynomial time*, M.Sc. thesis, Weizmann Institute of Science, Rehovot, Israel, 1988.

[26] A. Zamsky, *New algorithms for agreement in synchronous distributed networks*, M.Sc.

thesis, Technion—Israel Institute of Technology, 1992.

[27] A. ZAMSKY, *A Randomized Byzantine Agreement Protocol with Constant Expected Time and Guaranteed Termination in Optimal (Deterministic) Time*, in Proceedings of the 15th Annual ACM Symposium on Priniciples of Distributed Computing, 1996, pp. 201-208.

[28] A. ZAMSKY, A. ISRAELI AND S. PINTER, *Optimal time Byzantine agreement for* $t < n/8$, *Distributed Computing*, 9 (1995), pp. 95-108.