Byzantine Agreement with Optimal Early Stopping, Optimal Resilience and Polynomial Complexity

Ittai Abraham*
VMware Research
Palo Alto, CA, USA
iabraham@vmware.com

Danny Dolev†
Hebrew University of Jerusalem
Jerusalem, Israel
dolev@cs.huji.ac.il

ABSTRACT

We provide the first protocol that solves Byzantine agreement with optimal early stopping ($\min\{f+2, t+1\}$ rounds) and optimal resilience ($n > 3t$) using polynomial message size and computation.

All previous approaches obtained sub-optimal results and used resolve rules that looked only at the immediate children in the EIG (*Exponential Information Gathering*) tree. At the heart of our solution are new resolve rules that look at multiple layers of the EIG tree.

1. INTRODUCTION

In 1980 Pease, Shostak and Lamport [PSL80, LSP82] introduced the problem of Byzantine agreement, a fundamental problem in fault-tolerant distributed computing. In this problem n processes each have some initial value and the goal is to have all correct processes decide on some common value. The network is reliable and synchronous. If all correct processes start with the same initial value then this must be the common decision value, and otherwise the value should either be an initial value of one of the correct processes or some pre-defined default value.[1] This should be done in spite of at most t corrupt processes that can behave arbitrarily (called Byzantine processes). Byzantine agreement abstracts one of the core difficulties in distributed computing and secure multi-party computation — that of coordinating a joint decision. Pease et al. [PSL80] prove that Byzantine agreement cannot be solved for $n \leq 3t$. Therefore we say that a protocol that solves Byzantine agreement for $n > 3t$ has *optimal resilience*. Fisher and Lynch [FL82] prove that any protocol that solves Byzantine agreement must have an execution that runs for $t+1$ rounds. Dolev et al. [DRS90] prove that any protocol must have executions that run for $\min\{f+2, t+1\}$ rounds, where f is the actual number of corrupt processes. Therefore we say that a protocol that solves Byzantine agreement with $\min\{f+2, t+1\}$ rounds has *optimal early stopping*.

The protocol of [PSL80] has optimal resilience and optimal worst case $t+1$ rounds. However the message complexity of their protocol is exponential. Following this result, many have studied the question of obtaining a protocol with optimal resilience and optimal worst case rounds that uses only polynomial-sized messages (and computation).

Dolev and Strong [DS82] obtained the first polynomial protocol with optimal resilience. The problem of obtaining a protocol with optimal resilience, optimal worst case rounds and polynomial-sized messages turned out to be surprisingly challenging. Building on a long sequence of results, Berman and Garay [BG93] presented a protocol with optimal worst case rounds and polynomial-sized messages for $n > 4t$. In an exceptional tour de force, Garay and Moses [GM93, GM98], presented a protocol for binary-valued Byzantine agreement obtaining optimal resilience, polynomial-sized messages and $\min\{f+5, t+1\}$ rounds. We refer the reader to [GM98] for a detailed and full account of the related work. Recently Kowalski and Mostéfaoui [KM13] improved the message complexity to $\tilde{O}(n^3)$ but their solution does not provide early stopping and requires exponential computation.

Worst case running of $t+1$ rounds is the best possible if the protocol is to be resilient to an adversary that controls t processes. However, in executions where the adversary controls only $f < t$ processes, the optimal worst case can be improved to $f+2$ rounds. Berman et al. [BGP92] were the first to obtain optimal resilience and optimal early stopping (i.e. $\min\{f+2, t+1\}$ rounds) using exponential size messages. Early stopping is an extremely desirable property in real world replication systems. In fact, agreement in a small number of rounds when $f = 0$ is a core advantage of several practical state machine replication protocols (for example [CL99] and [KAD+07] focus on optimizing early stopping in the fault free case).

Somewhat surprisingly, after more than 30 years of research on Byzantine Agreement, the problem of obtaining the best of all worlds is still open. There is no protocol with

*Part of the work was done at Microsoft Research Silicon Valley.

†Part of the work was done while visiting Microsoft Research Silicon Valley. Danny Dolev is Incumbent of the Berthold Badler Chair in Computer Science. This research project was supported in part by The Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11), and by grant 3/9778 of the Israeli Ministry of Science and Technology.

[1]Other versions of the problem may not restrict to a value on one of the correct processes, if not all initial values are the same, or require agreement on a leader's initial value, which can be reduced to the version we defined.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
STOC'15, June 14–17, 2015, Portland, Oregon, USA.
Copyright © 2015 ACM 978-1-4503-3536-2/15/06 ...$15.00.
http://dx.doi.org/10.1145/2746539.2746581.

optimal resilience, optimal early stopping and polynomial-sized message. The conference version of [GM98] claimed to have solved this problem but the journal version only proves a $\min\{f+5, t+1\}$ round protocol, then says it is *possible* to obtain a $\min\{f+3, t+1\}$ round protocol and finally the authors say they *believe* it should be possible to obtain a $\min\{f+2, t+1\}$ round protocol. We could not see how to directly extend the approach of [GM98] to obtain optimal early stopping. The main contribution of this paper is solving this long standing open question and providing the optimal $\min\{f+2, t+1\}$ rounds with optimal resilience and polynomial complexity. Moreover, our result applies directly for arbitrary initial values and not only to binary initial values, as some of the previous results.

Our Byzantine agreement protocol obtains a stronger notion of *multi-valued validity*. If $v \neq \perp$ is the decision value then at least $t+1$ correct processes started with value v. The multi-valued validity property is crucial in our solution for early stopping with monitors. This property is also more suitable in proving that Byzantine agreement implements an ideal world centralized decider that uses the majority value. We note that several previous solutions (in particular [GM98]) are inherently binary and their extension to multi-valued agreement does not have the stronger multi-valued validity property.

THEOREM 1. *Given n processes, there exists a protocol that solves Byzantine agreement. The protocol is resilient to any Byzantine adversary of size $t < n/3$. For any such adversary, the total number of bits sent by any correct process is polynomial in n and the number of rounds is $\min\{f+2, t+1\}$ where f is the actual size of the adversary.*

Overview of our solution. At a high level we follow the framework set by Berman and Garay [BG93]. In this framework, if at a given round all processes seem to behave correctly then the protocol stops quickly thereafter. So if the adversary wants to cause the protocol to continue for many rounds it must have at least one corrupt process behave in a faulty manner in each round. However, behaving in a faulty manner will expose the process and in a few rounds the misbehaving process will become publicly exposed as corrupt.

This puts the adversary between a rock and a hard place: if too few corrupt processes are publicly exposed then the protocol reaches agreement quickly, if too many corrupt process are exposed then a "monitor" framework (also called "cloture votes") that runs in the background causes the protocol to reach agreement in a few rounds. So the only path the adversary can take in order to generate a long execution is to publicly expose exactly one corrupt process each round. In the $t < n/4$ case, this type of adversary behavior keeps the communication polynomial.

For $t < n/3$ a central challenge is that a corrupt process can cause communication to grow in round i but will be publicly exposed only in round $i+2$. Naively, such a corrupt process may also cause communication to grow both in round i and $i+1$ and this may cause exponential communication blowup. Garay and Moses [GM98] overcome this challenge by providing a protocol such that, if there are at most two new corrupt processes in round i and no new corrupt process in round $i+1$ then even though they are publicly exposed in round $i+2$ they cannot increase communication in round $i+1$ (also known as preventing "cross corruption").

At the core of the binary-valued protocol of Garay and Moses is the property that one value can only be decided on even rounds and the other only on odd rounds. This property seems to raise several unsolved challenges for obtaining optimal early stopping. We could not see how to overcome these challenges and obtain optimal early stopping using this property. Our approach allows values to be fixed in a way that is indifferent to the parity of the round number (and is not restricted to binary values).

Two key properties of our protocol that makes it quite different from all previous protocols. First, the value of a node is determined by the values of its children and grandchildren in the EIG tree ([BNDDS92]). Second, if agreement is reached on a node then the value of all its children is changed to be the value of the node. This second property is crucial because otherwise even though a node is fixed there could be disagreement about the value of its child. Since the value of the parent of the fixed node depends on its children and grandchildren, the disagreement on the grandchild may cause disagreement on the parent and this disagreement could propagate to the root.

The decision to change the value of the children when their parent is fixed is non-trivial. Consider the following scenario with a node σ, child σp and grandchild σpq: some correct reach agreement that the value of σpq is d, then some correct reach agreement that the value of σp is $d' \neq d$ and hence the value of σpq is changed (colored) to d'. So it may happen that some correct decide the value of σ based on σpq being fixed on d and some other correct decide the value of σ based on σpq being colored to d'. Making sure that agreement is reached in all such scenarios requires us to have a relatively complex set of complementary agreement rules.

To bound the size of the tree by a polynomial size we prove that the adversary is still between a rock and a hard place: roughly speaking there are three cases. If just one new process is publicly exposed in a given round then the tree grows mildly (remains polynomial). If three or more new processes are exposed in the same round then this increases the size of the tree but can happen at most a constant number of times before a monitor process will cause the protocol to stop quickly.

The remaining case is when exactly two new processes are exposed, then a sequence of (possibly zero) rounds where just one new process is exposed in each round, followed by a round where no new process is exposed. This is a generalized version of the "cross corruption" case of [GM98] where the adversary does not face increased risk of being caught by the monitor process. We prove that in these cases the tree essentially grows mildly (remains polynomial).

In order to deal with this generalized "cross corruption" we introduce a special resolve rule (SPECIAL-BOT RULE) tailored to this scenario. In particular, in some cases we fix the value of a node σ to $\perp$ (a special default value) if we detect enough support. This solves the generalized "cross corruption" problem but adds significant complications. Recall that when we fix a value to a node then we also fix (color) the children of this node with the same value.

Suppose a process fixes a node σ to $\perp$. The risk is that some correct processes may have used a child σp with value d but some other correct process will see $\perp$ for σp (because when σ is fixed to $\perp$ we color all its children to $\perp$). Roughly speaking, we overcome this difficulty by having two resolve rule thresholds. The base is the $n - t$ thresh-

old (RESOLVE RULE, IT-TO-RT RULE) and the other is with a $n-t-1$ threshold (RELAXED RULE). In essence this $n-t-1$ rule is resilient to disagreement on one child node (that may occur due to coloring). We then make sure that the SPECIAL-BOT RULE can indeed change only one child value. This delicate interplay between the resolve rules is at the core of our new approach.

The adversary. Given $n > 3t$ and $\phi \leq t$, as in [GM98], we will consider a (t, ϕ)-*adversary* - an adversary that can control up to ϕ corrupt processes that behave arbitrarily and at most $t - \phi$ corrupt processes that are always silent (send some default value $\perp$ to all processes every round). The (t, ϕ)-adversary will be useful to model executions in which all correct processes have detected beforehand some common set of at least $t-\phi$ corrupt processes and hence ignore them throughout the protocol. Note that the standard t-adversary is just a (t, t)-adversary.

2. THE EIG STRUCTURE AND RULES

In this section we define the EIG structure and rules.

Let N be the set of processes, $n = |N|$ and assume that $n > 3t$. Let D be a set of possible decision values. We assume some decision $\perp \in D$ is the designated default decision.

Let Σ_r be the set of all sequences of length r of elements of N without repetition. Let $\Sigma_0 = \epsilon$, the empty sequence. Let $\Sigma = \bigcup_{0 \leq j \leq t+1} \Sigma_j$. An *Exponential Information Gathering* tree (EIG in short) is a tree whose nodes are elements in Σ and whose edges connect each node to the node representing its longest proper prefix. Thus, node ϵ has n children, and a node from Σ_k has exactly $n - k$ children.

We will typically use the Greek letter σ to denote a sequence (possibly empty) of labels corresponding to a node in an EIG tree. We use the notation σq to denote the node in the EIG tree that corresponds to the child of node σ that corresponds to the sequence σ concatenated with $q \in N$. We denote by $\bar{\epsilon}$ the root node of the tree that corresponds to the empty sequence. Given two sequences $\sigma, \sigma' \in \Sigma$, let $\sigma' \sqsubset \sigma$ denote that σ' is a proper prefix of σ and $\sigma' \sqsubseteq \sigma$ denote that σ' is a prefix of σ (potentially $\sigma' = \sigma$).

In the EIG consensus protocol each process maintains a dynamic tree data structure $\mathcal{IT}$. This data structure maps a set of nodes in σ to values in D. Intuitively, this tree contains all the information the process has heard so far. Each process z also maintains two global dynamic sets $\mathcal{F}, \mathcal{FA}$. The set $\mathcal{F}$ contains processes that z detected as faulty, and $\mathcal{FA}$ contains processes that z knows are detected by all correct processes. The protocol for updating $\mathcal{F}, \mathcal{FA}$ is straightforward:

- In each round the processes exchange their $\mathcal{F}$ lists and update their $\mathcal{F}$ and $\mathcal{FA}$ sets once a faulty process appears in $t + 1$ or $2t + 1$ lists, respectively.
- When a process is detected as faulty every correct process masks its future messages to $\perp$.

The basic EIG protocol will be invoked repeatedly, and several copies of the EIG protocol may be running concurrently. The accumulated set of faulty processes will be used across all copies (the rest of the variables and data structures are local to each EIG invocation). Therefore, we assume that when the protocol is invoked the following property holds:

PROPERTY 1. *When the protocol is invoked, no correct process appears in the faulty sets of any other correct process. Moreover, $\mathcal{FA}_p \subseteq \mathcal{F}_p$ and $\mathcal{FA}_p \subseteq \mathcal{F}_q$ for any two correct processes p and q,*

Each invocation of the EIG protocol is tagged with a parameter ϕ, known to all processes. An EIG protocol with parameter ϕ, will run for at most $\phi+1$ rounds. At the beginning of the agreement protocol the faulty sets are empty at all correct processes and the EIG protocol with parameter $\phi = t$ is executed. Each additional invocation of the EIG protocol is with a smaller value of ϕ. In the non-trivial case, when the EIG protocol with parameter ϕ is invoked then $|\bigcap_i \mathcal{FA}_i| \geq t - \phi$. There will be one exception to this assumption, and it is handled in Lemma 1. Thus, other than in that specific case, it is assumed that we have a (t, ϕ)-adversary during the execution of the EIG protocol with parameter ϕ.

The basic EIG protocol for a correct process z with initial value $d_z \in D$ is very simple:

1. **Init:** Set $\mathcal{IT}(\bar{\epsilon}) := d_z$, so $\mathcal{IT}(\bar{\epsilon})$ is set to be the initial value.
2. **Send:** in each round r, $1 \leq r \leq \phi + 1$, for every $\sigma \in \mathcal{IT} \cap \Sigma_{r-1}$, such that $z \notin \sigma$, send the message $\langle \sigma, z, \mathcal{IT}(\sigma) \rangle$ to every process.
3. **Receive set:** in each round r, let $\mathcal{S}_r := \{\sigma x \in \Sigma_r\}$.
4. **Receive rule:** in each round r, for all $\sigma x \in \mathcal{S}_r$ set
$$\mathcal{IT}(\sigma x) := \begin{cases} \perp & \text{if } x \in \mathcal{F} \\ d & \text{if } x \notin \mathcal{F} \text{ sent } \langle \sigma, x, d \rangle \text{ and } d \in D; \\ \mathcal{IT}(\sigma) & \text{otherwise.} \end{cases}$$

Note: assigning of $\mathcal{IT}(\sigma x) := \mathcal{IT}(\sigma)$ when $x \notin \mathcal{F}$ is crucial for the case where x is correct and has halted in the previous round. Thus, if a process is silent but is not detected (possibly because it has halted due to early stopping) z assigns it the value it heard in the previous round.

We use a second dynamic EIG tree data structure $\mathcal{RT}$. Intuitively, if a process puts a value in a node of this tree then, essentially, all correct processes will put the same value in the same node in at most 2 more rounds. Processes use several rules to close branches of the $\mathcal{IT}$ tree whose value in $\mathcal{RT}$ is already determined by all. We present later the rules for closing branches of the $\mathcal{IT}$ tree. To handle this, we modify lines 2 and 3 as described below (and keep lines 1 and 4 as above).

2. **Send:** in each round r, $1 \leq r \leq \phi + 1$, for every $\sigma \in \mathcal{IT} \cap \Sigma_{r-1}$, such that $z \notin \sigma$, and the branch σ is not closed send the message $\langle \sigma, z, \mathcal{IT}(\sigma) \rangle$ to every process.
3. **Receive set:** in each round r, let $\mathcal{S}_r = \{\sigma x \in \Sigma_r \mid$ branch σx is not closed$\}$.

Informally, $\mathcal{IT}_z(\sigma p) = d$ (where $\mathcal{IT}_z$ denotes the $\mathcal{IT}$ tree at process z) indicates that process z received a message from process p that said that his value for σ was d. $\mathcal{RT}_z(\sigma p) = d$ indicates, essentially, that process z knows that every correct process x will agree and have $d \in \mathcal{RT}_x(\sigma p)$ in at most two more rounds.

Observe that we record in the EIG tree only information from sequences of nodes that do not contain repetition, therefore, not every message a process receives will be recorded.

At the end of each round, we apply the rules below to determine whether to assign values to nodes in $\mathcal{RT}$, assigning that value in $\mathcal{RT}$ is called *resolving* the node.

2.1 The Resolve Rules

A key feature of our algorithm is that whenever we put a value into $\mathcal{RT}(\sigma)$ we also color (assign) all the descendants of σ in $\mathcal{RT}$ with the same value. Observe that this means we may color a node σw in $\mathcal{RT}$ to d even if w is correct and sent $d' \neq d$ to all other correct processes.

Rules for IT-to-RT resolve: The following definitions and rules cause a node to be resolved based on information in $\mathcal{IT}$.

1. If $\mathcal{IT}(\sigma w) = d$ **then** we say: (1) w is a voter of (σ, w, d); (2) w is confirmed on (σ, w, d); (3) For all $v \in N \setminus \{\sigma\}$, w is a supporter of v on (σ, w, d).

   Note: the reason that we count w as a voter, as confirmed and as a supporter for all its echoers is that due to the EIG structure w does not appear in the subtree of σw.

2. If $\mathcal{IT}(\sigma wv) = d$, **then** we say that v is a supporter of v for (σ, w, d).

   Note: again we need v to be a supporter of itself because of the EIG structure.

3. If $\mathcal{IT}(\sigma wvu) = d$ **then** we say that u is a supporter of v for (σ, w, d).

4. If there is a set $|U| = n - t$, such that for each $u' \in U$, u' is a supporter of v on (σ, w, d) **then** we say that v is confirmed on (σ, w, d).

   Note: if σ contains no correct and w is correct, then any correct child v (of σw) will indeed have $n - t$ supporters for σw and hence will be confirmed. Note that one supporter is w, the other is v and the remaining are all the $n - t - 2$ correct children of σwv. Also note that w is confirmed, so all $n - t$ correct will be confirmed on (σ, w, d).

5. If $u \neq w$ has a set $|V| = n - t$, such that for each $v' \in V$, u is a supporter of v' on (σ, w, d) and v' is confirmed on (σ, w, d) **then** u is a voter of (σ, w, d).

   Note: this is somewhat similar to the notion of a Voter in grade-cast ([FM97, FM88]). But there is a crucial difference: all the $n - t$ echoers need to be *confirmed*. Also note that w is a voter for itself.

6. IT-TO-RT RULE: If w has a set $|U| = n - t$, such that for each $u' \in U$, u' is a voter of (σ, w, d) **then** if $\sigma w \notin \mathcal{RT}$, then put $\mathcal{RT}(\sigma w) := d$ and color descendants of σw with d as well.

   Note: this is somewhat similar to the notion of a grade 2 in grade-cast. A crucial difference is that the $n - t$ voters needed are defined with respect to *supported* echoers. This is a non-trivial change that breaks the standard grade-cast properties. Also note that we not only put a value in σw but also color all the descendants.

7. ROUND $\phi + 1$ RULE: if $\mathcal{IT}(\sigma w) = d$ and $\sigma \in \Sigma_t$ **then** if $\sigma w \notin \mathcal{RT}$, then put $\mathcal{RT}(\sigma w) := d$.

   Note: this is a standard rule to deal with the last round.

Rules for $\mathcal{RT}$ tree resolve: The following definitions and rules cause a node to be resolved based only on information in $\mathcal{RT}$ (these rules do not look at $\mathcal{IT}$).

1. If there is a set $|U| = t + 1$, such that for each $u' \in U$, $\mathcal{RT}(\sigma wvu') = d$ **then** we say v is $\mathcal{RT}$-confirmed on (σ, w, d).

   Note: if any correct sees a node as confirmed then it has $n - t$ that echo its value. At least $t + 1$ of them are correct and they all cause all correct to see the node as $\mathcal{RT}$-confirmed. Of course a node may become $\mathcal{RT}$-confirmed even if it was never confirmed by any correct. Observe that if $\mathcal{RT}(\sigma wu) = d$ then, by coloring, u is $\mathcal{RT}$-confirmed on (σ, w, d).

2. If $u \neq w$ has a set $|V| = n - t$, such that each $v' \in V$ is $\mathcal{RT}$-confirmed on (σ, w, d) and for each $v' \in V \setminus \{u\}$, $\mathcal{RT}(\sigma wv'u) = d$ and if $u \in V$ then also $\mathcal{RT}(\sigma wu) = d$, **then** u is $\mathcal{RT}$-voter of (σ, w, d).

   Note: if any correct process sees a node as a voter then it has $n - t$ echoers that are confirmed. So each of these $n - t$ echoers will be $\mathcal{RT}$-confirmed. So all correct processes will see this node as $\mathcal{RT}$-voter. Of course a node can become $\mathcal{RT}$-voter even if it was never a voter at any correct process.

3. RESOLVE RULE: If w has a set $|U| = t + 1$, such that for each $u' \in U$, u' is a $\mathcal{RT}$-voter of (σ, w, d) **then** if $\sigma w \notin \mathcal{RT}$, then put $\mathcal{RT}(\sigma w) := d$, and color descendants of σw with d as well. The rule applies also for node $\sigma w = \bar{\epsilon}$.

   Note: if any correct process does IT-TO-RT RULE then this rule tries to guarantee that all correct processes will also put this node in $\mathcal{RT}$. The problem is that SPECIAL-BOT RULE (see below) may be applied to one of the echoers and this may cause some of the $\mathcal{RT}$-voters to lose their required support. The following rule fixes this situation. It reduces the threshold to $n - t - 1$ but requires that all children nodes are fixed.

4. RELAXED RULE: If all the children of σw are in $\mathcal{RT}$ (*i.e.,*$\forall \sigma wv \in \Sigma$: $\sigma wv \in \mathcal{RT}$) and exists a set $|V| = n - t - 1$, such that for each $v' \in V$, $\mathcal{RT}(\sigma wv') = d$, **then** if $\sigma w \notin \mathcal{RT}$, then put $\mathcal{RT}(\sigma w) := d$, and color descendants of σw with d as well. The rule applies only for nodes $|\sigma w| \geq 1$.

   Note: as mentioned above, the RELAXED RULE requires a threshold of $n - t - 1$ so that it can take into account the possibility of one value changing to $\perp$ due to the following rule:

5. SPECIAL-BOT RULE: If there is a set $|V| = t + 2 - |\sigma wu|$ such that for all $v \in V$, $\mathcal{RT}(\sigma wuv) = \perp$ and for all $u' \neq u$ such that $\sigma wu' \in \Sigma$, $\sigma wu' \in \mathcal{RT}$ **then** if $\sigma wu \notin \mathcal{RT}$, then put $\mathcal{RT}(\sigma wu) := \perp$, and color descendants of σwu with $\perp$ as well. The rule applies only for $|\sigma wu| \geq 2$.

   Note: This rule can be applied to at most one child.

6. SPECIAL-ROOT-BOT RULE: If exists a set $|U| = t+1$ such that for each $u \in U$, $\mathcal{RT}(u) = \perp$ then if $\bar{\epsilon} \notin \mathcal{RT}$, then put $\mathcal{RT}(\bar{\epsilon}) := \perp$, and color descendants of $\bar{\epsilon}$ with $\perp$ as well.

   Note: this rule is important in order to stop quickly if $t+1$ correct processes start with the value $\perp$.

To prevent the data structures from expanding too much processes close branches of the tree, and from that point on they do not send messages related to the closed branches. We use the notation $\{\sigma \in \mathcal{RT}[r]\}$ to denote an indicator variable that equals true if $\mathcal{RT}(\sigma)$ was assigned some value by the end of round r, and false otherwise.

Branch Closing and Early Resolve rules: There are three rules to close a branch in $\mathcal{IT}$ two of them also trigger an early resolve. By the end of round r, $r \leq \phi$,

1. DECAY RULE: if $\exists \sigma' \sqsubseteq \sigma$ such that $\sigma' \in \mathcal{RT}[r-1]$, then close the branch $\sigma \in \mathcal{IT}$.

   Note: this is the simple case: if a process already fixed the value of σ' in $\mathcal{RT}$ in round $r-1$ then it stops in the end of round r, since by the end of round $r+1$ all correct processes will put σ' in $\mathcal{RT}$ (and will interpret this process's silence in the right way during round $r+1$). There is no need to continue. Coloring will fix all the values of this subtree.

2. EARLY_IT-TO-RT RULE: if $\sigma \in \Sigma_{r-1}$ and exists $U \subseteq N$, $U \cap \{u' \mid u' \in \sigma\} = \emptyset$, $|U| = n - r$, such that for every $u, v \in U \setminus \mathcal{F}$, $\mathcal{IT}(\sigma u) = \mathcal{IT}(\sigma v)$, then if $\sigma \notin \mathcal{RT}$, then put $\mathcal{RT}(\sigma) := \mathcal{IT}(\sigma)$ and close the branch $\sigma \in \mathcal{IT}$.

   Note: this is a case where the process can forecast that all correct processes will put σ in $\mathcal{RT}$ in the next round (because the process sees that all children nodes agree). So the process can fix σ in this round and stop now, because all correct processes will fix σ in $\mathcal{RT}$ next round (and will interpret this process's silence in the right way).

3. STRONG_IT-TO-RT RULE: if $\sigma \in \Sigma_{r-2}$ and exists $U \subseteq N$, $U \cap \{u' \mid u' \in \sigma\} = \emptyset$, $|U| = n - r + 1$ such that for every $u, v \in U \setminus \mathcal{F}$, where $v \neq u$, $\mathcal{IT}(\sigma uv) = \mathcal{IT}(\sigma vu)$ then, if $\sigma \notin \mathcal{RT}$, then put $\mathcal{RT}(\sigma) := \mathcal{IT}(\sigma)$ and close the branch $\sigma \in \mathcal{IT}$.

   Note: in this case all the correct children of σ except for at most one will be fixed in the next round to the same value, so the RELAXED RULE will be applied to σ in the next round. So we can fix σ in this round and stop now.

In each round all the above rules are applied repeatedly until none holds any more.

The rules above imply that there are two ways to give a value to a node in $\mathcal{RT}$. One is assigning it a value using the various rules, and the other is coloring it as a result of assigning a value to one of its predecessors. We will use the term *color* for the second one and the term *put* for the first one.

Rules for fault detection and masking: The following definitions and rules are used to detect faulty processes, put them into $\mathcal{F}$ and hence mask them (all messages from $\mathcal{F}$ are masked to $\perp$). The last rule also defines an additional masking. The process first updates its $\mathcal{F}$ and $\mathcal{FA}$ sets using the sets received from the other processes during the current round. A process is added to $\mathcal{F}$ or $\mathcal{FA}$ once it appears in $t+1$ or $2t+1$ sets, respectively. Next the process applies the following fault detection rules. The fault detection is executed before applying any of the resolve rules above. When a new process is added to $\mathcal{F}$, the new masking is applied and the fault detection is repeated until no new process can be added. Only then the resolve rules above are applied.

At process z by the end of round r:

1. **Not Voter**: If $\exists \sigma w \in \Sigma_{r-1}$ and $w \neq z$ and $\nexists \sigma' \sqsubseteq \sigma w$ such that $\sigma' \in \mathcal{RT}$ and it is not the case that there exists a set $|U| = n - t - 1$ such that for each $u' \in U$, $\mathcal{IT}(\sigma w u') = \mathcal{IT}(\sigma w)$ **then** add w to $\mathcal{F}$.

   Note: this is the standard detection rule after one round - if anything looks suspicions then detect.

2. **Not IT-to-RT**: If $\exists \sigma w \in \Sigma_{r-2}$ for which w does not have a set $|U| = n - t$, such that for each $u' \in U$, u' is a voter of (σ, w, d), and $\nexists \sigma' \sqsubseteq \sigma$ such that $\sigma' \in \mathcal{RT}$ **then** add w to $\mathcal{F}$.

   Note: this is the standard detection rule after two rounds - if anything looks suspicions then detect.

3. If u, $u \neq w$, has a set $|V| = n - t$, such that for each $v' \in V$, u is a supporter of v' on (σ, w, d) **then** we say that u is an *unconfirmed voter* of (σ, w, d).

   Note: the notion of an *unconfirmed voter* is exactly that of a voter in the standard grade-cast protocol.

4. If w has a set $|U| = t+1$, such that for each $u' \in U$, u' is an unconfirmed voter of (σ, w, d) **then** we say that σw is *leaning towards* d.

   Note: the notion of *leaning towards* is exactly that of getting grade ≥ 1 in the standard grade-cast protocol.

5. **Not Masking**: If $\sigma w \in \Sigma_{r-3}$ is leaning towards d and there exists u, $|V| = t+1$, and $d' \neq d$ such that for each $v' \in V$, $\mathcal{IT}(\sigma w u v') = d'$ and there exists $|\sigma''| > |\sigma|$ such that $\mathcal{IT}(\sigma'' w u) \neq \perp$ then
   (a) $\mathcal{IT}(\sigma'' w u) = \perp$;
   (b) if by the end of the round $\nexists \sigma' \sqsubseteq \sigma'' w$ such that $\sigma' \in \mathcal{RT}$ **then** add u to $\mathcal{F}$.

   Note: If σw is leaning towards d then u must have heard at least $t+1$ say d on σw. If $t+1$ say u said d' then u must have said d' to some correct. So u must have received d' from σw but in the next round u hears $t+1$ say σw said d. So u must conclude that w is faulty and u must mask him from the next round. If u did not mask some $\sigma'' w u$ then the **Not Masking** rule will detect u as faulty and mask all such $\sigma'' w u$ for you and also mark you as faulty. The reason we wait until the end of the round to add that node to $\mathcal{F}$ is that it might be a node of a correct process that stopped in the previous round and hence did not send any messages in the current round, and therefore did not send masking. In such a case we mask its virtual sending, but do not add it to $\mathcal{F}$.

Finalized Output: By the end of each round (after applying all the resolve rules), the process checks whether there

is a frontier in $\mathcal{RT}$. A *frontier* (also called a cut) is said to exist if for all $\sigma \in \Sigma_{\phi+1}$ there exists some sub-sequence $\sigma' \sqsubseteq \sigma$ such that $\sigma' \in \mathcal{RT}$.

1. **Early Output rule**: By the end of a round, if $\bar{\epsilon} \in \mathcal{RT}$, **output** $\mathcal{RT}(\bar{\epsilon})$.
2. **Final Output rule**: Otherwise, if there is a frontier, **output** $\perp$.

Observe that the existence of a frontier can be tested from the current $\mathcal{IT}$ in $O(|\mathcal{IT}|)$ time.

Stopping rule: If all branches of IT are closed, stop the protocol.

3. THE CONSENSUS PROTOCOL ANALYSIS

The EIG protocol implicitly presented in the previous section is a consensus protocol $\mathcal{D}_\phi$, where ϕ, $1 \leq \phi \leq t$ is a parameter. Protocol $\mathcal{D}_\phi$ runs for at most $\phi + 1$ rounds and solves Byzantine agreement against a (t, ϕ)-adversary. Denote by G the set of correct processes, $|G| \geq n - t$, where $n = |N|$, and by S, $S = \bigcap_{q \in G} \mathcal{F}_q$, the set of processes that are masked to $\perp$ by all correct processes. Let $s := |S|$.

Our solution invokes several copies of the EIG protocol. For each invoked protocol, $\mathcal{D}_\phi$, there are two cases: either $s \geq t - \phi$, or we are guaranteed that the input of all correct processes that start the protocol is the same (in particular, it may be that some correct processes have halted and do not start the protocol). The following lemma deals with this latter case.

LEMMA 1. *[Validity and Fast Termination] For any (t, t)-adversary, and $n \geq 3t + 1$,*

1. *if every correct process that starts the protocol holds the same input value d then d is the output value of all correct processes that start the protocol, by the end of round 2, and all of them complete the protocol by the end of round 3.*
2. *if all correct processes start the protocol and $t + 1$ correct processes start with $\perp$ then all correct processes output $\perp$ by the end of round 3 and stop the protocol by the end of round 4.*
3. *For $p, q \in G$, no p will add q to $\mathcal{F}_p$ in either of the above cases.*

The only case in which not all correct processes invoke a $\mathcal{D}_\phi$ protocol is when some of the background running monitors are being invoked by some of the correct processes, while others may have already stopped. This special case is guaranteed to be when the inputs of all participating correct processes is $\perp$, and consensus can be still be achieved. Lemma 1 implies the following:

COROLLARY 1. *For any (t, t)-adversary, and $n \geq 3t + 1$, if every correct process that invokes the protocol start with input $\perp$, then $\perp$ is the output value at each participating correct process by the end of round 2, and each participating correct process completes the protocol by the end of round 3. Moreover, for $p, q \in G$, no p will add q to $\mathcal{F}_p$.*

The gossip exchange among correct processes about identified faults ensures the following:

LEMMA 2. *For a (t, ϕ)-adversary and protocol $\mathcal{D}_\phi$, $n \geq 3t + 1$, assuming Property 1, for any k, $1 \leq k \leq \phi + 1$, by the end of round k, for every two correct processes p, q, $\mathcal{FA}_p \subseteq \mathcal{F}_q$ and $\mathcal{FA}_p[k-1] \subseteq \mathcal{FA}_p[k]$.*

A node may initially assign a value using one of the "put" rules and later it may color it to a different value. In the arguments below we sometimes need to refer to the value that was put to a node rather than the value it might be colored to. Once a node has a value it is not assigned a value using any put rule any more. Thus, the value assigned using a put rule is an initial value that may be assigned to a node before it is colored, or that node may never have a value put to it. To focus on these put operations, we will add, for proof purposes, that whenever a node p uses a put rule for some σ, except ROUND $\phi + 1$ RULE, it also puts σ in $\mathcal{PT}_p$ (The "Put-Tree") and as a result at that moment, $\mathcal{PT}_p(\sigma) = \mathcal{RT}_p(\sigma)$. We do not color nodes in $\mathcal{PT}_p$, thus for σ that is colored, but was not assigned a value prior to that, $\mathcal{PT}_p(\sigma)$ is undefined. We exclude ROUND $\phi + 1$ RULE from $\mathcal{PT}$ on purpose.

The following is the core statement of the technical properties of the protocol. The only way we found to prove all these is via an induction argument that proves all properties together. The theorem contains four items.

The detection part proves that correct processes are never suspected as faulty. The challenge is that the various rules instruct processes when to stop sending messages, and that might cause other correct processes to be suspected as faulty.

The validity part proves that if a correct process sends a value, it will reach the $\mathcal{RT}$ of every other correct process within two rounds. It also proves that if a correct process decides not to send a value (thus, closed a branch), the appropriate node will be in $\mathcal{RT}$ of every correct process. The third claim in the validity part is that if a process appears in $\mathcal{FA}$, then it appears in $\mathcal{RT}$ of every correct process within two rounds.

The safety part intends to prove consistency in the $\mathcal{RT}$. The challenge is that coloring may cause the trees of correct processes to defer. Therefore the careful statements looks at $\mathcal{PT}$, and which rule was used in order to assign the value to it. The $\perp$ value is a default value, therefore there is a special consideration of whether the value the process puts is $\perp$ or not. The end result is that if a node appears in $\mathcal{PT}$ of two correct processes, it carries the same value.

The liveness part shows that if a node appears in $\mathcal{RT}$ of a correct process, it will appear in $\mathcal{RT}$ of any other correct process within two rounds.

THEOREM 2. *For a (t, ϕ)-adversary and protocol $\mathcal{D}_\phi$, $n \geq 3t + 1$, assuming Property 1 and that all correct processes participate in the protocol, then for any $1 \leq k \leq \phi + 1$:*

1. **No False Detection:** *For $p, q \in G$, no q will add p to $\mathcal{F}_q$ in round k.*
2. **Validity:**
   (a) *For $\sigma \in \Sigma_{k-3}$ if $p \in G$, sends $\langle \sigma, p, d_p \rangle$, then at the end of round k, at every correct process x, either $\mathcal{RT}_x(\sigma p) = d_p$ or $\exists \sigma' \sqsubset \sigma$ such that $\sigma' \in \mathcal{RT}_x$. For $k = \phi + 1$, the property holds also for any $\sigma \in \Sigma_{k-2}$ and for any $\sigma \in \Sigma_{k-1}$.*
   (b) *If $z \in \mathcal{FA}$ in the beginning of round $k-2$, then by the end of round k, at every correct process, either $\mathcal{RT}(\sigma z) = \perp$ or $\exists \sigma' \sqsubset \sigma$ such that $\sigma' \in \mathcal{RT}$. For*

$k = \phi + 1$, *the property holds for* $z \in \mathcal{FA}$ *in the beginning of rounds* $k-1$ *or* k.

(c) *For* $\sigma \in \Sigma_{k-1}$, *if* $p \in G$, *does not send* $\langle \sigma, p, d \rangle$ *for any* $d \in D$, *then at the end of round* k, *at every correct process* x, $\exists \sigma' \sqsubset \sigma$ *such that* $\sigma' \in \mathcal{RT}_x$.

3. **Safety:** *For* $p, q \in G$, $x \in N$, $|\sigma x| \leq \phi$,$\sigma x \in \mathcal{PT}_p[k]$, *then*

(a) *if* p *applies* RESOLVE RULE *to put* $\mathcal{PT}_p(\sigma x) = d$, $d \neq \perp$, *and* v *is one of the* $\mathcal{RT}$*-confirmed nodes on* (σ, x, d) *in* $\mathcal{RT}_p$ *used in applying this rule in* $\mathcal{RT}_p$, *and in addition* $\mathcal{PT}_q(\sigma x v) = \perp$, *then* q *applied* SPECIAL-BOT RULE *to put* $\sigma x v$;

(b) *if* $|\sigma x| \geq 1$ *and* $\mathcal{PT}_p(\sigma x) = d$, $d \neq \perp$, *then, by the end of round* k, $|V_q| \leq t$, *where* $V_q = \{u \mid \mathcal{PT}_q(\sigma x u) = \perp\}$;

(c) *if* $|\sigma x| \geq 1$ *and* $\mathcal{PT}_p(\sigma x) = \perp$ *and it wasn't put using* SPECIAL-BOT RULE, *then, by the end of round* k, $|V_q| \leq t$, *where* $V_q = \{u \mid \mathcal{PT}_q(\sigma x u) \neq \perp\}$;

(d) *if* $\sigma x \in \mathcal{PT}_q[k]$, *then* $\mathcal{PT}_p(\sigma x) = \mathcal{PT}_q(\sigma x)$.

4. **Liveness:** *For* $p, q \in G$, *if* $\sigma \in \mathcal{RT}_p[k-2]$ *then* $\sigma \in \mathcal{RT}_q[k]$. *For* $k = \phi + 1$, *if* $\sigma \in \mathcal{RT}_p$ *then* $\sigma \in \mathcal{RT}_q$.

The following Theorem summarizes the properties needed from our protocol.

THEOREM 3. *For a* (t, ϕ)*-adversary and protocol* $\mathcal{D}_\phi$ *and* $n \geq 3t+1$ *and assuming that all correct processes participate in the protocol:*

1. *Every correct process outputs the same value.*
2. *If the input values of all correct processes are the same, this is the output value. Every correct process outputs it by round* 2 *and stops by round* 3.
3. *If* $t+1$ *of the correct processes hold an input value of* $\perp$, *then all correct processes output* $\perp$ *by the end of round* 3 *and stop by the end of round* 4.
4. *If the actual number of faults is* $f_\phi < \phi$, *then all correct processes complete the protocol by the end of round* $f_\phi + 2$.
5. *If the actual number of faults is* $f_\phi = 0$, *and all correct processes start with the same initial value, then all correct processes complete the protocol by the end of round* 1.
6. *If the actual number of faults is* $f_\phi = 1$, *and all correct processes start with the same initial value, then all correct processes complete the protocol by the end of round* 2.
7. *If a correct process outputs in round* k, *it stops by the end of round* $k+1$.
8. *If a correct process stops in the end of round* k, *all correct processes output by round* $k+1$ *and stop by round* $k+2$.

4. MONITORS

We follow the approach of [BG93, GM93, GM98] with some modifications for guaranteeing early stopping.

In round $r = 1$ we run $\mathcal{D}_t$ using the initial values. For each integer k, in round $1 < r = 1 + 4k < t - 1$ we invoke protocol $\mathcal{D}_{t-1-4k}$ whose initial values is either $\perp$ (meaning everything is OK) or BAD (meaning that too many corrupt processes were detected). We call this sequence of protocols the *basic monitor sequence*. We will actually run 4 such sequences.

4.1 The Basic Monitor Protocol

Each process z stores two variables: $v \in D$, the current value, and *early*, a boolean value. Initially v equals the initial input of process z and $early := false$. Later, $early = true$ will be an indicator that the next decision protocol must decide $\perp$ (because there is not enough support for BAD). Each process remembers the last value of $early_q$ it received from every other process q, even if q did not send one recently.

Throughout this section we use the notation: $\bar{r} \equiv r \pmod 4$.

Algorithm 1: The Basic Monitor protocol (at process z)

```
1:  if r̄ = 1:
2:      if r < t − 1 then invoke protocol D_{t+1−r} with
        initial value v_z;
3:  if r̄ = 2:
4:      at the end of the round:
5:          if |FA| ≥ r + 3 then set v_z := BAD
6:              otherwise set v_z := ⊥;
7:  if r̄ = 3:
8:      send v_z to all;
9:      at the end of the round:
10:         if |{q | v_q = BAD}| ≤ t then set early_z := true
11:             otherwise set early_z := false;
12: if r̄ = 0:
13:     send early_z to all;
14:     at the end of the round:
15:         if |{q | early_q = true}| ≥ t+1 then set v_z := ⊥;
16:         if every previously invoked protocol produced
        an output then set v_z := ⊥.
```

The monitor protocol runs in the background until the process halts. The monitor protocol invokes a new $\mathcal{D}_\phi$ protocol every 4 rounds. In each round, the monitor's lines of code are executed before running all the other protocols, and its end of round lines of code are executed before ending the current round in all currently running protocols. This is important, since it needs to detect, for example, whether all currently running protocols produced outputs for determining its variable for the next round. At the end of each round the monitor protocol applies the monitor_halting and monitor_decision rules below to determine whether to halt all the running protocols at once, or only to commit to the final decision value.

When a process is instructed to apply a monitor_decision it applies the following definition. If it is instructed to halt (monitor_halting), then if it did not previously apply the monitor_decision, it applies monitor_decision first and then halts all currently running protocols that were invoked by the monitor at once.

DEFINITION 1 (MONITOR_DECISION). *A process that did not previously decide,* **decides** BAD, *if any previously invoked protocol outputs* BAD. *Otherwise, it decides on the output of* $\mathcal{D}_t$.

When a process is instructed to decide without halting, it may need to continue running all protocols for few more rounds to help others to decide. We define "halt by $r + x$" to mean continue to run all active protocols until the end of round $\min\{r + x, t + 1\}$, unless an halt is issued earlier.

4.2 Monitor Halting and Decision Conditions

Given that different processes may end various invocations of the protocols in different rounds we need a rule to make sure that all running protocols end by the end of round $f+2$. The challenge in stopping all protocols by the end of $f+2$ is the fact that individual protocols may end at round $f+2$ and we do not have a room to exchange extra messages among the processes. This also implies that we need to have a halting rule at every round of the monitor protocol, since $f+2$ may occur at any round.

Each halting rule implies how other rules need to be enforced in later rounds, since any process may be the first to apply a monitor_halting at a given round and we need to ensure that for every extension of the protocols, until everyone decides, all will reach the same decision despite the fact that those that have halted are not participating any more. The conditions take into account processes that may have halted. A process considers another one as halted if it doesn't receive any message from it in any of the concurrently running set of invoked protocols, monitors and the gossiping of $\mathcal{F}$.

To achieve that we add the following set of rules.

Monitor Halting Rules:

H_{BAD}. Apply monitor_halting if any monitor stops with output BAD. Otherwise if any monitor outputs BAD, apply monitor_decision now and monitor_halting by $r+2$.

H_1. Case $\bar{r}=1$:

(a) If all previously invoked protocols stopped, apply monitor_halting.

(b) Otherwise, if only the latest invoked protocol did not stop and $|\{q \mid early_q = true \text{ or } q \text{ halted}\}| \geq n-t$, then apply monitor_halting.

(c) Otherwise, if only the latest invoked protocol did not stop and $|\{q \mid early_q = true \text{ or } q \text{ halted}\}| \geq t+1$, then apply monitor_decision now and monitor_halting by $r+2$.

H_2. Case $\bar{r}=2$:

(a) If all previously invoked protocols stopped, apply monitor_halting.

(b) Otherwise, if only the latest invoked protocol did not stop and $|\{q \mid early_q = true \text{ or } q \text{ halted}\}| \geq n-t$ was true in the previous round, then apply monitor_halting.

(c) Otherwise, if only the latest invoked protocol did not stop and $|\{q \mid early_q = true \text{ or } q \text{ halted}\}| \geq t+1$ was true in the previous round, then apply monitor_decision and now and monitor_halting by $r+1$.

H_3. Case $\bar{r}=3$: If all previously invoked protocols stopped, apply monitor_halting.

H_4. Case $\bar{r}=0$: If all previously invoked protocols stopped and $|\{q \mid early_q = true \text{ or } q \text{ halted}\}| \geq n-t$ then apply monitor_halting.

LEMMA 3. *If $n > 3t$ and there are f, $f \leq t$, corrupt processes then all correct processes apply monitor_halting by the end of round* $\min(t+1, f+2)$.

LEMMA 4. *If the first process applies monitor_halting in round r on d then every correct process applies monitor_decision by round* $\min\{r+4, f+2, t+1\}$, *applies monitor_halting by round* $\min\{r+5, f+2, t+1\}$, *and obtains the same decision value, d.*

Lemma 3 and 4 complete the correctness part of Theorem 1. To simplify the polynomial considerations we look at a pipeline of monitors.

4.3 Monitors Pipeline

The basic monitor protocol runs a sequence of monitors and tests the number of faults' threshold every 4 rounds (Line 5). This allows the adversary to expose more faults in the following round, and be able to further expand the tree before the threshold is noticed the next time the processes execute Line 5. To circumvent this we will run a pipeline of 3 additional sequences of monitors on top of the basic one appearing above. Doing this we obtain that in every round r one of the 4 monitor sequences will be testing the threshold on the number of faults

Monitor sequence i, for $1 \leq i \leq 4$ begins in round i and invokes protocols every 4 rounds, in every round r, $1 < r = i + 4k < t-1$, it invokes protocol $\mathcal{D}_{t-i-4k}$. Monitor sequence 1 is the basic monitor sequence defined in the previous subsection. Each monitor sequence independently runs the basic monitor protocol (Figure 1) every 4 rounds. In the monitor protocol, the test $\bar{r}=j$, which stands for $\bar{r} \equiv r \pmod 4$ in the basic monitor sequence, is replaced with $\bar{r_i}=j$, which stands for $\bar{r_i} \equiv r+1-i \pmod 4 = j$ (naturally only for $r+1-i>0$). Each of the four monitor sequences decides and halts separately, as in the previous section above.

Notice that protocol $\mathcal{D}_t$ is invoked only by the basic sequence (Sequence 1). For each of the three other monitor sequences, the decision rule is: decide BAD, if any invoked protocol (in this sequence) outputs BAD, and $\perp$ otherwise. Observe that Lemma 3 and 4 hold for each individual sequence.

We now state the global decision and global halting rules:

DEFINITION 2 (GLOBAL HALTING). *If any monitor sequence halts with* BAD, *or all 4 monitor sequences halt, the process halts.*

DEFINITION 3. *The global_decision is the output of $\mathcal{D}_t$, unless any monitor sequence returns* BAD, *in which case the decision is* BAD.

The following are immediate consequences of Lemma 3 and 4 and the above definitions.

COROLLARY 2. *If $n > 3t$ and there are f, $f \leq t$, corrupt processes then all correct processes halt by the end of round* $\min(t+1, f+2)$.

COROLLARY 3. *If the first correct process halts in round r on d then every correct process applies global_decision by round* $\min\{r+4, f+2, t+1\}$, *halts by round* $\min\{r+5, f+2, t+1\}$, *and obtains the same decision value.*

5. BOUNDING THE SIZE OF THE TREE

Following the approach of [GM98], we make the following definitions:

DEFINITION 4. *A node $\sigma z \in \Sigma$ is* fully corrupt *if there does not exist $p \in G$ and $\sigma' \sqsupseteq \sigma z$ such that $\sigma' \in \mathcal{RT}_p[|\sigma z|+2]$.*

DEFINITION 5. *We say that* a process z becomes fully corrupt at i *if exists a node $\sigma z \in \Sigma$ that is fully corrupt, $|\sigma z| = i$*

and for every previous node $|\sigma' z| < i$, node $\sigma' z$ is not fully corrupt.

The following is immediate from the definitions above.

CLAIM 1. *If process z becomes fully corrupt at i then of all the nodes of Σ that end with z only nodes of round i and $i+1$ can be fully corrupt.*

PROOF. By definition of fully corrupt, all correct processes will have $z \in \mathcal{F}$ in round $i+2$. So in that round and later all nodes will put $\perp$ in $\mathcal{RT}$ for z. □

Let $\mathcal{CT}$, the *corrupt tree*, be a dynamic tree structure. $\mathcal{CT}$ is the tree of all fully corrupt nodes (note that due to coloring, the set of fully corrupt nodes is indeed a tree). We denote by $\mathcal{CT}[i]$ the state of $\mathcal{CT}$ at the end of round i. By the definition of fully corrupt, at round i we add nodes of length $i-2$ to $\mathcal{CT}$.

We *label* the nodes in $\mathcal{CT}$ as follows: a node $\sigma z \in \mathcal{CT}$ is a *regular* node if process z becomes fully corrupt at $|\sigma z|$ and $\sigma z \in \mathcal{CT}$ is a *special* node if process z becomes fully corrupt at $|\sigma z| - 1$.

Let α_i denote the distinct number of processes that become fully corrupt at round i. For convenience, define $\alpha_0 = 0$ (this technicality is useful in Lemma 7). Let $A = \alpha_0, \alpha_1, \ldots$ be the sequence of counts of process that become fully corrupt in a given execution.

Following the approach of [GM98], we define $waste_i = (\sum_{j \le i} \alpha_i) - i$. So $waste_i$ is the number of processes that became fully corrupt till round i minus i (the round number). The following claim connects $waste_i$ to $\cap_{p \in G} \mathcal{FA}[i+3]_p$ the set of fully detected corrupt processes at round $i+3$.

CLAIM 2. *For any round $4 \le r \le t+1$, and any correct process we have $|\mathcal{FA}[r]| \ge \sum_{j \le r-3} \alpha_i$.*

PROOF. By the definition of z becoming fully corrupt at i, all correct processes will have $z \in \mathcal{F}$ in round $i+2$. Due to the gossiping of $\mathcal{F}$, all correct processes will have $z \in \mathcal{FA}$ in round $i+3$. □

So if $waste_i \ge 6$ then in round $r = i+3$ we will have $\left(\sum_{j \le i} \alpha_i\right) - i \ge 6$ so by Lemma 2 for each correct process we have $|\mathcal{FA}[r]| \ge r+3$. In this case all correct processes will start in the associated monitor sequence the next protocol with initial value BAD and the protocol and monitor sequence and global protocol will reach agreement and halt on BAD by round $i+6$ (by Lemma 1).

We will now show that if the adversary maintains a small waste (less than 6 by the argument above, but this will work for any constant) then the $\mathcal{CT}$ tree must remain polynomial sized.

The following key lemma shows that the adversary cannot increase the number of leaves by "cross contamination". In more detail, if the adversary causes two fully corrupt processes at round i_1 followed by a sequence of rounds with exactly one fully corrupt process at each round followed by a round with no fully corrupt process at that round then this action essentially keeps the tree $\mathcal{CT}$ growing at a slow (polynomial) rate. We note that the focus on "cross contamination" follows the approach of [GM98]. But they only verify the case of two fully corrupt followed by a round with no fully corrupt. We have identified a larger family of adversary behavior that does not increase the waste (in the long run). Our proof covers this larger set of behaviors and this requires additional work.

LEMMA 5. *Assume $0 < i_1 < i_2$ such that $\alpha_{i_1} = 2$, $\alpha_{i_2} = 0$ and for all $i_1 < i < i_2$, $\alpha_i = 1$ then for any $\sigma \in \Sigma_{i_1-1} \cap \mathcal{CT}$ it is not the case that there exists $\sigma p \tau \in \Sigma_{i_2+1} \cap \mathcal{CT}$ and there exists $\sigma q \tau \in \Sigma_{i_2+1} \cap \mathcal{CT}$ (so there is at most one extension). Moreover the size of the subtree starting from σp or σq and ending in length i_2+1 is bounded by $O((i_2 - i_1)^2)$.*

See the additional analysis in Section 5.1.

To bound the size of $\mathcal{CT}$, we partition the sequence $A = \alpha_0, \alpha_1, \ldots$ by iteratively marking subsequences using the following procedure. For each subsequence we mark, we prove that it either causes the tree to grow in a controllable manner (so the ending tree is polynomial), or it causes the tree to grow considerably (by a factor of $O(n)$) but at the price of increasing the waste by some positive constant. Since the waste is bounded by a constant, the result follows.

1. By Lemma 7 we know that if A contains a $0(1)^*0$ (a sequence starting with 0 then some 1's then 0) then it contains it just once as a suffix of A. Moreover, this suffix does not increase the size of the tree by more than $O(n)$. Let A_1 be the resulting unmarked sequence after marking such a suffix (if it exists).
2. Mark all subsequences in A_1 of the form $2(1)^*0$ (a sequence starting with 2 then some 1's then 0). By Lemma 5 each such occurrence will not increase the number of leafs in $\mathcal{CT}$ (but may add branches that will close whose total size is at most n^2 over all such sequences). Let A_2 be the remaining unmarked subsequences.
3. Mark all subsequences in A_2 of the form $X(1)^*0$ where $X \in \{3, \ldots, t\}$ (a sequence starting with 3 or a larger number followed by some 1's then 0). By Lemma 8 each occurrence of such a sequence may increase the size of the tree multiplicatively by $O(n)$ leafs and $O(n^2)$ non-leaf nodes, but this also increases the *waste* by $c - 1 > 1$ (where c is the first element of the subsequence). Observe that the remaining unmarked subsequences do not contain any element that equals 0. Let A_3 be the remaining unmarked subsequences.
4. Mark all subsequences of the form $Y(1)^*$ where $Y \in \{2, \ldots, t\}$ (a sequence whose first element is 2 or a larger number followed by some 1's but no zero at the end). Again, by Lemma 8 each such occurrence may increase the size of the tree by $O(n)$ leafs and $O(n^2)$ non-leafs, but this also increases the *waste* by $c > 1$. Let A_4 be the remaining unmarked.
5. Since A_3 contains no element that equals zero and we removed all subsequences that have element of value 2 or larger as the first element then A_4 must either be empty or A_4 is a prefix of A of the form $(1)^*$ (a series of 1's). Since it is a prefix of A then a sequence of 1's keeps at most one leaf. So the tree remains small.

Thus, the size of $\mathcal{CT}$ is polynomial, which by Lemma 6 bounds the size of $\mathcal{IT}$. This completes the proof of Theorem 1.

5.1 Additional Analysis

The following lemma bounds the size of $\mathcal{IT}$ as a function of the size of $\mathcal{CT}$ times $O(n^7)$.

LEMMA 6. *If $\sigma \in \mathcal{IT}$ and $|\sigma| > 7$ then there exists $\sigma' \sqsubset \sigma$ with $|\sigma'| \ge |\sigma| - 7$ such that $\sigma' \in \mathcal{CT}$.*

The following lemma shows that the protocol stops early if the adversary causes two rounds with no new fully corrupt and only one fully corrupt per round between them.

LEMMA 7. *If exists* $0 \leq i_1 < i_2$ *such that* $\alpha_{i_1} = 0$, $\alpha_{i_2} = 0$ *and for all* $i_1 < i < i_2$, $\alpha_i = 1$ *then all processes will halt by the end of round* $i_2 + 5$.

The following lemma shows that having a large number (3 or more) of processes becoming fully corrupt at a given round, followed by a sequence of 1's and then maybe followed by 0 does increase the number of leafs considerably. Note that if $\alpha_{i_1-1} + \alpha_{i_1} \geq 6$ then the monitor process will cause the protocol to reach agreement and stop in a constant number of rounds. So we only look at the case that $\alpha_{i_1-1} + \alpha_{i_1} < 6$.

LEMMA 8. *If* $2 < \alpha_{i_1}$, $\alpha_{i_1-1} + \alpha_{i_1} < 6$, $\alpha_{i_2} \in \{0, 1\}$ *and for all* $i_1 < i < i_2$, $\alpha_i = 1$ *then for any* $\sigma \in \Sigma_{i_1-1} \cap \mathcal{CT}$ *there are at most* $O(i_2 - i_1)$ *nodes of the form* $\sigma\tau \in \Sigma_{i_2+1} \cap \mathcal{CT}$. *Moreover the size of the subtree starting from* σ *and ending in length* $i_2 + 1$ *is bounded by* $O((i_2 - i_1)^2)$.

6. CONCLUSION

In this paper we resolve the problem of the existence of a protocol with polynomial complexity and optimal early stopping and resilience. The main remaining open question is reducing the complexity of such protocols to a low degree polynomial. Another interesting open problem is obtaining unbeatable protocols [CGM14] (which is a stronger notion than early stopping).

We would like to thank Yoram Moses and Juan Garay for insightful discussions and comments.

7. REFERENCES

[BG93] Piotr Berman and Juan A. Garay. Cloture votes: n/4-resilient distributed consensus in t+1 rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993.

[BGP92] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal early stopping in distributed consensus. In Adrian Segall and Shmuel Zaks, editors, *Distributed Algorithms*, volume 647 of *Lecture Notes in Computer Science*, pages 221–237. Springer Berlin Heidelberg, 1992.

[BNDDS92] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: changing algorithms on the fly to expedite byzantine agreement. *Inf. Comput.*, 97:205–233, April 1992.

[CGM14] Armando Castañeda, Yannai A. Gonczarowski, and Yoram Moses. Unbeatable consensus. In Fabian Kuhn, editor, *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2014.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the third symposium on Operating systems design and implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[DRS90] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37:720–741, October 1990.

[DS82] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *ACM Symposium on Theory of Computing*, pages 401–407, New York, NY, USA, 1982. ACM.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *ACM Symposium on Theory of Computing*, pages 148–161, 1988.

[FM97] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[GM93] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement in t + 1 rounds. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 31–41, New York, NY, USA, 1993. ACM.

[GM98] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for processors in rounds. *SIAM J. Comput.*, 27:247–290, February 1998.

[KAD+07] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 45–58, New York, NY, USA, 2007. ACM.

[KM13] Dariusz R. Kowalski and Achour Mostéfaoui. Synchronous byzantine agreement with nearly a cubic number of communication bits. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 84–91, New York, NY, USA, 2013. ACM.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401, July 1982.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.