Optimal Early Stopping in Distributed Consensus

(Extended Abstract)

Piotr Berman[1] Juan A. Garay[2,3] Kenneth J. Perry[3]

[1] Department of Computer Science
The Pennsylvania State University
University Park, PA 16802, USA
[2] Dept. of Applied Math and Computer Science
The Weizmann Institute of Science
Rehovot, 76100 Israel
[3] IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA

Abstract. The *Distributed Consensus* problem involves n processors each of which holds an initial binary value. At most t processors may be faulty and ignore any protocol (even behaving maliciously), yet it is required that the non-faulty processors eventually agree on a value that was initially held by one of them. This paper presents consensus protocols that tolerate arbitrary faults, are *early-stopping* (i.e., run for a number of rounds proportional to the number of faults f that actually occur during their execution), and are optimal in various measures.

Our first contribution is an early-stopping consensus protocol that is simultaneously optimal in round complexity (i.e., $min(t+1, f+2)$ rounds) and number of processors (i.e., $n > 3t$). This settles a long-standing open question [DRS, 1982]. These bounds were not known to be attainable even with the use of authentication. Since consensus is not attainable with $n \leq 3t$ we provide a definitive answer to the problem of early-stopping consensus. Previous protocols for this problem that achieved round optimality required $n = \Omega(t^2)$ [C, DRS], $n > 6t$ [MW] and $n > 4t$ [BGP]. Instrumental in achieving this result is the new *safe message reconstruction* technique, which we expect to be of broader applicability.

The previous protocol is round- and processor-optimal, but not efficient. Our second contribution is a pair of optimal early-stopping consensus protocols that use messages of *constant* size. No previously existing early-stopping protocol, whether exactly or only asymptotically optimal in some measures, has used messages of constant size. The only other existing early-stopping protocol with constant-sized messages requires $n = \Omega(t^2)$ [C].

Finally, we indicate how to extend one of the previous protocols to be optimal in total bit (and message) complexity and number of processors. This is the first protocol that is both optimal in these measures and early-stopping.

1 Introduction and Problem Statement

There are many situations in the management of distributed systems with one common characteristic: a collection of processors must coordinate a decision. This problem becomes non-trivial when some of the processors are faulty and cannot be relied upon to faithfully obey a protocol. It is reasonable to at least demand that, regardless of the behavior of the faulty processors, correct processors can reach an agreement consistent with the initial value of a correct processor. The *Distributed Consensus* problem (and its twin, *Byzantine Agreement*) [LSP] provides an abstract setting in which methods for tolerating faults may be explored and perhaps influence practical designs. The problem can be formally stated as follows.

We are given a set of processors P of which some unknown subset T, $\#T \leq t$, are *faulty* and may exhibit arbitrary (*Byzantine*) behavior. Processors from $P-T$ are called *correct*. Every processor is given an initial value, 0 or 1. After the execution of the protocol, the final values of the correct processors have to satisfy the following two conditions, regardless of the behavior of the faulty processors:

- *Agreement*: they are all equal.
- *Validity*: they are all equal to the initial value, if the latter is unique.

We use the standard model of a synchronous network of processors numbered from 1 to n. The computation performed by the network evolves as a series of *rounds*, during which the processors send messages, receive them and perform local computations according to the protocol.

There already exist many protocols for this problem because there are several quality parameters that can be optimized: the total number of processors n (as a function of t), the number of rounds of communication r, and the communication complexity given by, e.g., the maximal message size m. Typically, some parameters are optimized to the detriment of the others, but there is no proof that all parameters cannot be made optimal simultaneously.

A lower bound of $t+1$ rounds of communication has been established for this problem even if no failures actually occur [DS]. Thus, overcoming potential faults is expensive, even though, in practice, they are rarely observed. This has led others to explore ways in which the price of fault tolerance may be reduced. Hadzilacos and Halpern [HH] have recently concentrated on runs that are failure-free; they present protocols with optimal number of messages for these runs. Dolev, Reischuk, and Strong [DRS] are concerned with protocols that take a number of rounds proportional to f, the number of failures that actually occur, rather than t, the worst-case number of failures. By permitting processors to stop at different times (instead of simultaneously) they show that *early-stopping* protocols are possible and demonstrate a lower bound of $min(f+2, t+1)$ rounds. Randomization is another way of circumventing the $t+1$ lower bound on round complexity. In this setting, protocols that run for an expected constant number of rounds are possible. Feldman and Micali [FM], in particular, have a randomized consensus protocol that achieves optimal number of processors and polynomial message size.

The contributions of this paper are deterministic early-stopping consensus protocols that are optimal in various measures. Our first contribution is the first consensus protocol to simultaneously achieve optimal early stopping (i.e., $r = min(t+1, f+2)$) and optimal number of processors (i.e., $n > 3t$ [LSP]). This settles a question posed by Dolev *et al.* in 1982 [DRS]. These (non-authenticated) bounds were not known to be attainable even when the faulty processors are restricted not to corrupt a given authentication scheme.[4] Previous protocols with optimal early stopping were sub-optimal in processors, namely, they required $n = O(t^2)$ [C, DRS], $n > 6t$ [MW] and $n > 4t$ [BGP]. Instrumental in achieving this result is the new *safe message reconstruction* technique, which we expect to be of broader applicability.

Our protocol is optimal in number of processors and number of rounds, but not in message size. Our second contribution is a pair of early-stopping consensus protocols that use messages of *constant* size. One of our protocols uses single-bit messages, is optimal in the number of processors, and asymptotically optimal in the number of rounds. No other early-stopping protocol [BGP, DRS, TPS, W], whether exactly or only asymptotically optimal in some measures, has used messages of constant size. To the best of our knowledge, the only other existing early-stopping protocol with constant-sized messages is also sub-optimal in processors, i.e., it requires $n = O(t^2)$ [C].

Finally, we present an early-stopping consensus protocol that is asymptotically optimal in the total number of bits transfered ($O(nt)$ [DR]) and the number of processors. This is the first protocol that is both optimal in these measures and early-stopping. A consensus protocol with optimal bit complexity is, *a fortiori*, optimal with respect to message complexity. Hadzilacos and Halpern have recently achieved consensus with (exact) message optimality, but only in the case of runs that are *failure-free* [HH]. The rationale is that since failures are hopefully rare, optimality in the usual case is to be emphasized. However, in case (even few) failures do occur, their solution requires either a large round overhead (if asymptotical message optimality is to be maintained), or a loss of $\Omega(n)$ in message complexity (if early termination is important). The consensus protocol we present here achieves message optimality in all runs, at the expense of a reasonably small constant and is early-stopping.

The remainder of the paper is organized as follows. In Section 2 we present the early-stopping consensus protocol that is optimal in both the number of processors and the number of rounds. In Section 3 we present the protocols that use messages of constant size, while in Section 4 we indicate how optimal bit (and message) complexity can be achieved, without sacrificing the early stopping property. Sections 3 and 4 use completely different techniques from those of Section 2, thus admitting independent readings.

[4] For the more restricted failure classes (crash, omission), the round and the corresponding processor lower bounds ($n > t + 1$) are known to be simultaneously attainable ([LF], [PT], respectively).

2 Early Stopping in the Minimum Number of Rounds

The early-stopping consensus protocol of this section is presented as the result of a sequence of transformations to the *Exponential Information Gathering (EIG)* protocol (a refinement of the original agreement protocol of Lamport *et al.* [LSP] due to Bar-Noy *et al.* [BDDS]). First we introduce the notion of *safe decisions* and the rules to obtain them. These rules by themselves allow us to re-derive a protocol first presented by Waarts [W] which omits certain messages of EIG and reduces the number of rounds to $min(t+1, f+3)$. Then we introduce the notion of *safe message reconstruction*, and show that a correct processor may stop sending messages after it decides because other correct processors can safely infer the omitted messages. The subtle part is that although the outcome of safe message reconstruction does not necessarily agree with the suppressed messages, it nevertheless leads to the same final decision. The safe decision rules and safe message reconstruction technique, when combined with the EIG protocol, allow us to prove our first result: an early-stopping consensus protocol that is optimal in both the number of processors and the number of rounds, i.e., $r = min(t+1, f+2)$.

In this section we assume that $n = 2p + t - 1$, where $p > t$ (equivalently, $n > 3t$ and $n - t$ is odd). We use $|\sigma|$ to denote the length of string σ.

2.1 EIG Overview

We first present a short synopsis of protocol EIG. Processors build trees of the following form: the set of nodes consists of sequences from P^* without repetitions, and of length at most $t+1$; the root is λ (the empty sequence); the children of σ, $EXT(\sigma)$, are nodes of the form σj. In each tree, a node σ has a *value* denoted $Val(\sigma)$. Each correct processor initializes a one-node tree consisting of λ and sets $Val(\lambda)$ to its initial value. Then, for rounds $0, 1, \cdots, t$ correct processor i applies the following two primitives to all leaves of its tree:

- Send(σ): send $Val(\sigma)$ to all processors (provided i does not appear in σ).
- Receive(σ, j): upon receiving v when j executes Send(σ), create node σj and set $Val(\sigma j) = v$.

Once the tree is constructed, the processors use the rules given below to compute predicates $Dec(\sigma, v)$ and $PDec(\sigma, v)$ for each node σ and $v \in \{0,1\}$ (we read $Dec(\sigma, v)$ ($PDec(\sigma, v)$) as "value v is the decision (a partial decision) for σ"):

$$PDec(\sigma, v) \;\equiv\; \begin{cases} Val(\sigma) = v & \text{if } \sigma \text{ is a leaf,} \\ \#\{\tau \in EXT(\sigma) : Dec(\tau, v)\} \geq p & \text{otherwise;} \end{cases}$$

$$Dec(\sigma, v) \;\equiv\; PDec(\sigma, v) \wedge \neg PDec(\sigma, \overline{v}).$$

The final value for the consensus problem is 1 iff $Dec(\lambda, 1)$. Note that $Dec(\sigma, v)$ implies $\neg Dec(\sigma, \overline{v})$, although it is possible that $\neg Dec(\sigma, 0) \;\wedge\; \neg Dec(\sigma, 1)$ holds.

Definition 1. A predicate π at node σ is *common* iff π is true at σ in the tree of every correct processor. A node σ is common iff either $\neg Dec(\sigma, 0) \wedge \neg Dec(\sigma, 1)$ is common, or for some v, $Dec(\sigma, v)$ is common.

The following two lemmas express the correctness of EIG in terms useful for our further discussion. Lemma 2 implies the Validity condition, while Lemma 3 implies Agreement.

Lemma 2. *Assume that either*

- $\sigma = \lambda$ *and the initial value of every correct processor is* v, *or*
- $\sigma = \tau i$, *in* i*'s tree* $Val(\tau) = v$ *and* i *executes* $Send(\tau)$ *correctly.*

Then $Dec(\sigma, v)$ *is common.*

The proof follows directly from the definition by induction on $|\sigma|$; the base case is $|\sigma| = t$ and the induction parameter is decreasing.

Lemma 3. *The root of the* EIG *tree,* λ, *is common.*

Proof. (Sketch) By an induction similar to the one in the previous lemma, we prove that if a common node exists on every path from σ to a leaf, then σ is common. Note that every leaf has an ancestor of the form σi, where i is correct and, as a consequence of Lemma 2, node σi is common. □

2.2 Safe Rules for Early Stopping

In the protocols presented later in this section the values of predicates are actually computed differently and *not always consistently* with EIG. The following definition and lemmas provide justification for such deviations. Ideally, every processor would reach the same decision as in EIG for every node. However, we will show that it suffices to reach *safe* decisions for a well-defined class of nodes.

Definition 4. A predicate π is *safe* if either

- π is common, or
- π refers to a node $\sigma \notin T^*(P-T)^*$, or
- π is of the form $Dec(\sigma j, v)$ and the predicates $PDec(\sigma, v)$ and $PDec(\sigma j, v)$ are both common.

Note that the agreement value is 1 iff $Dec(\lambda, 1)$ is safe (i.e., iff $Dec(\lambda, 1)$ is common).

We now present a set of *Early Stopping Rules* that enable a processor to determine when "partial-decision" and "decision" predicates are safe. These rules can be used in a number of ways:

- In *EIG*, the rules are only used to determine what to decide, i.e., which "decision" predicate is safe for the root.

– In *Early-Stopping EIG* (the protocol of Theorem 8) rules 1-7 are additionally used to prevent the building of the entire tree, so that a process can stop early. That is, when "decision" predicates for a node are safe there is no need to continue determining the nodes descendants.
– Our contribution is to use *knowledge* about "decision" and "partial-decision" predicates in order to reason about what the descendants of a node *should be* without actually having to receive messages relating to them. We call this technique *safe message reconstruction*, and describe it in subsection 2.3. We call the resulting protocol *Safe EIG*.

The first set of rules is applied by each processor in order to calculate when "partial-decision" predicates are safe.

1. If in the tree of a correct processor σ is created in round t and $Val(\sigma) = v$, then $PDec(\sigma, v)$ and $\neg PDec(\sigma, \overline{v})$ are safe.
2. If in the tree of a correct processor $\#\{\tau \in EXT(\sigma) : Val(\tau) = v\} \geq p + t - |\sigma|$, then $PDec(\sigma, v)$ is safe.
3. If in the tree of a correct processor $\#\{\tau \in EXT(\sigma) : Val(\tau) = v\} \geq p + 2 \cdot (t - |\sigma|)$, then $PDec(\sigma, v)$ and $\neg PDec(\sigma, \overline{v})$ are safe.
4. If $\#\{\tau \in EXT(\sigma) : Dec(\tau, v)$ is safe $\} \geq p$, then $PDec(\sigma, v)$ is safe.
5. If $\#\{\tau \in EXT(\sigma) : PDec(\tau, \overline{v})$ is safe $\} \geq p + t - |\sigma|$, then $\neg PDec(\sigma, v)$ is safe.

The next set of rules allows a processor to calculate when "decision" predicates are safe, given safe "partial-decision" predicates.

6. If $PDec(\sigma, v)$ and $\neg PDec(\sigma, \overline{v})$ are safe, then $Dec(\sigma, v)$ is safe.
7. If $PDec(\sigma, v)$ and $PDec(\sigma, \overline{v})$ are safe, then $\neg Dec(\sigma, v)$ and $\neg Dec(\sigma, \overline{v})$ are safe.
8. If $PDec(\sigma, v)$ and $PDec(\sigma j, v)$ are safe, then $Dec(\sigma j, v)$ is safe.

When a "decision" predicate is determined to be true for a node σ of processor i's tree, we say that i *decides* on σ. We say that i *decides promptly* on σ whenever it is able to decide through the application of rules (3) and (6).

The proof that the rules are sound is omitted from the abstract but follows from an easy induction and the following case analysis: If $\sigma \notin T^*(P - T)^*$ the claim is vacuous—any predicate is safe; if $\sigma \in T^*(P - T)^+$ then we can apply Lemma 2; lastly, if $\sigma \in T^*$, then at most $t - |\sigma|$ elements of $EXT(\sigma)$ store values received from faulty processors.

The following example illustrates how rules and decisions work.

Example 1. We illustrate how "prompt decisions" are reached; processors reach "decisions" in a similar manner. Assume $t = 2$ (hence $p = 3$) and $n = 7$. Figure 1 shows processor 2's tree after the second round (recall that since $r = t + 1$, the depth of the final *EIG* tree in this case equals 3). In the example, processor $4 \in P - T$ and $6, 7 \in T$. Given that 5 of the children of node 4 store the same value (0), and $p + 2 \cdot (t - |\sigma|) = 5$, for $|\sigma| = |4| = 1$, rule 3 "fires." This in turn makes rule 6 to fire, allowing processor 2 to decide promptly that 0 is a safe value for node 4. We remark that, despite processor 2 already having decided on node 4, traditional thinking still requires the processor to continue to report values (in this case, regarding node 4's children), since those values might be needed by other correct processors in order to reach their own decisions. In subsection 2.3 we show how this can be avoided.

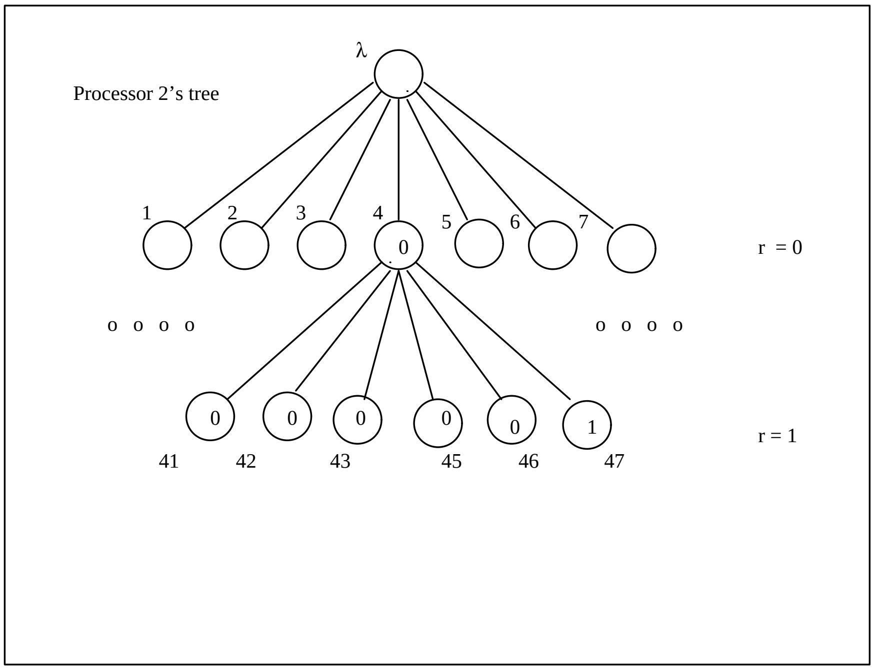

Fig. 1. How early stopping rules and decisions work.

When discussing the conclusions that may be derived from the above Early Stopping Rules, we use notation from the theory of knowledge in distributed systems [HM]. We read $K_i^r\phi$ as "after receiving round r's messages processor i knows fact ϕ." We will also use the "everybody knows" operator $E^r\phi \equiv \bigwedge_{i\in P-T} K_i^r\phi$. We will drop i and r when obvious from the context. It is assumed that the Early Stopping Rules are applied by all the correct processors, and that all their knowledge is the one that may be derived by applying these rules. We also assume that processors do not forget, i.e. $K_i^r\phi$ implies $K_i^{r+1}\phi$. A statement of the form $\neg K_i^r\phi$ means that i cannot deduce ϕ by round r by applying the rules.

We begin by stating some properties about the level of knowledge concerning safe predicates. The next two lemmas, whose proofs are omitted from the abstract, are analagous to Lemmas 2 and 3 and show that the Early Stopping

Rules are sufficient to infer the consensus value.

Lemma 5. *Assume that processor i in round r executes Send(σ) correctly, and in its tree $Val(\sigma) = v$. Then $E^{r+2}[Dec(\sigma i, v)$ is safe].*

Lemma 6. *Assume that after round r processor i has decided on an ancestor of every leaf node. Then i can also decide on λ.*

Let f denote the number of processors that actually fail and let the initial round of the protocol be numbered 0. Now we can show that by round $f+1$ all correct processors decide on λ. This happens because every leaf has an ancestor in $T^*(P-T)$. If a node of this form is created before round f, then all correct processors decide on it before round $f+2$; if it is created at round f, then all of its children are created by correct processors and consequently all correct processors decide on it promptly in round $f+1$.

Lemma 7. *Let $i \in P-T$ and $\sigma \in T^*(P-T)^*$. Then $K_i^r[Dec(\sigma, v)$ is safe] implies $E^{r+2}[Dec(\sigma, v)$ is safe]. Moreover, if i decides promptly on σ in round r, then $E^{r+1}[Dec(\sigma, v)$ is safe].*

The above lemma shows that processors not only can decide early, but that they may stop sending messages early. This justifies *Early-Stopping EIG*, a modified *EIG* protocol that enables early stopping by omitting the sending of messages as follows: a processor does not execute Send($\sigma\tau$), for any τ, if it has decided on σ before the previous round, or if it has decided promptly on σ in the previous round. "Decided" here means according to Early Stopping Rules (1-7). This allows us to prove the following theorem, originally due to Waarts [W]:

Theorem 8. Early-Stopping EIG *solves the Distributed Consensus problem using $n > 3t$ processors and $min(t+1, f+3)$ communication rounds.*

2.3 Optimal Early Stopping

Using a different formalism and only Early Stopping Rules (1-7), we have thus far described and proved correct a version of *EIG* that stops in $\min(t+1, f+3)$. Now the task is to show that starting from round $f+2$ the correct processors may stop sending messages, so that only rounds 0 to $f+1$ are needed. Achieving this with the optimal number of processors settles a long-standing open question [DRS].

As shown above, processors decide on λ by round $f+1$ at the latest; however this is not yet a sufficient reason to stop sending messages. Existing early-stopping protocols operate on the principle that if a correct processor knows that no other correct processor will read its message, it does not send it. In other words, a processor does not send any messages if it knows that all correct processors know the consensus value. However, it can be shown that when $n < 4t$ this level of knowledge cannot be assured by round $f+1$.

One can offer a different principle: a processor need not send a message if other processors can infer it from its silence. For example, if processor i does not send anything in round 1, it means that i decided on λ promptly; this may happen only if all messages i received in round 0 were the same. In general, such a complete reconstruction is not possible if the silence always immediately follows the round in which i decided on λ. However, we will show that while an *accurate* reconstruction is not possible, a *safe* reconstruction is.

Given some rules for message reconstruction, silence is equivalent to sending a message which is an alteration of a message in the EIG protocol. Such an alteration may *affect* some nodes in the sense that some correct processors may infer different predicates concerning these nodes. We say that an alteration of a message is *safe* if λ is not affected. When a processor i decides not to send some values and another processor j computes the safe alterations of these values, we say that j *safely reconstructs* these values.

We will use a very simple principle for not sending messages: processor i does not execute Send($\sigma\tau$) iff it already decided on σ. Other processors can proceed in three stages: reconstructing σ, reconstructing i's decision on σ, and reconstructing i's messages on the descendants of σ.

The reconstruction of σ is easy: first, all descendants of σ are omitted from the message; second, either $\sigma = \lambda$, or σ has a sibling with a descendant for which i did execute a Send (this is true because when a processor decides on all the children of a node, then it also decides on the node itself).

The reconstruction of decisions uses the following definitions. Let us fix one correct processor, say j, and let U be the set of nodes for which processor i executed Send before round r.

Definition 9. For every $\tau \in U$

$$\begin{aligned} CDec(i,\tau,v) &\equiv \neg K_j^{r-1}[\neg K_i^{r-1}[Dec(\tau,v) \text{ is safe}]]; \\ CPDec(i,\tau,v) &\equiv \neg K_j^{r-1}[\neg K_i^{r-1}[PDec(\tau,v) \text{ is safe}]]. \end{aligned}$$

In other words, $CDec(i,\tau,v)$ (resp., $CPDec(i,\tau,v)$) means that the assertion "$Dec(\tau,v)$ (resp., $PDec(\tau,v)$) may be inferred by i before round r" is consistent with the messages received by j before round r.

Lemma 10. *$CDec(i,\tau,v)$ implies $\neg CPDec(i,\tau,\overline{v})$ for every $\tau \in U$, $v = 0,1$.*

The proof of this lemma follows by induction on the depth of the subtree of U rooted by τ. The basis is proved by the analysis of prompt decisions, the inductive case by the analysis of the remaining safe rules. Rule (8) is needed in one of the cases.

The general principle for reconstructing a message is also simple: the reconstructed values will "maximally help" to reach the same decision on σ as i did. If i is faulty, then the reconstruction does not matter since a faulty processor is allowed to send anything. If i is correct, then the affected nodes must be proper descendants of σ, which makes the reconstruction safe. The implementation of this principle is described in the following lemma.

Lemma 11. *Processor j can safely reconstruct the values of Send's that i did not execute in round r.*

Proof. (Sketch) First, j finds those σ's which had to be decided on by i. Then for each such σ, processor j finds which of the following three cases holds: $CDec(i, \sigma, 0)$, $CDec(i, \sigma, 1)$ or $CPDec(i, \sigma, 0) \wedge CPDec(i, \sigma, 1)$; by Lemma 10 these cases are mutually exclusive. The safe reconstruction is as follows: in the first case, messages on all descendants of σ are reconstructed to 0; the second case is analogous. In the third case we can show that a reconstruction is safe if it gives v to a node of the form $\sigma k \tau$ whenever $CDec(i, \sigma k, v)$ is true. The reconstruction of the messages for the descendants of any σ not covered by this rule is irrelevant. □

The following example illustrates the technique.

Example 2. Assume the same scenario as in Example 1 ($t = 2$, $p = 3$, $n = 7$). Figure 2(a) shows processor 2's tree after the third (and last) round, during which no messages from processor 5 are received regarding node 4's children. We note again that we can assume that processor 5 is non-faulty. Processor 2 is able to establish $CDec(5, 4, 0)$ by reasoning as follows. There are two cases:

1. $4 \in P - T$. Since the number of "reported" 0's for node 4 is four, processor 5 must have decided promptly on 0, since it is not possible for processor 5's tree to have five 1's reported for 4.
2. $4 \in T$. Since $t \leq 2$, there remains at most one other faulty processor, which means there can be at most one different value in the children of node 4 in processor 5's tree. This in turn implies that 5 must have received at least five 0's to be able to decide promptly.

Processor 2 then proceeds (Lemma 11) to reconstruct missing messages from processor 5 on all descendants of node 4 to 0 (see Figure 2(b)). We note that in the process some values of nodes might be affected (e.g., nodes 465 and 475 in Figure 2(b)), but this deviation won't propagate above the node under consideration (node 4 in the example).

For space considerations, the previous example illustrated the simplest case of message reconstruction originating from a prompt decision (processor 5's). We remark that the other cases—non prompt decision, "neutral" decision—are more involved, but the case analyses in the reconstruction of messages and decisions are similar and mostly reduce. We leave the details for the full version of the paper.

This justifies the protocol we call *Safe EIG*, similar to *EIG* except for the following:

- A processor executes Send($\sigma\tau$), for any τ, iff it has not decided on σ ("decided" here means derivable from Early Stopping Rules (1-8)), and
- missing messages that are needed are reconstructed according to the safe reconstruction technique of Lemma 11.

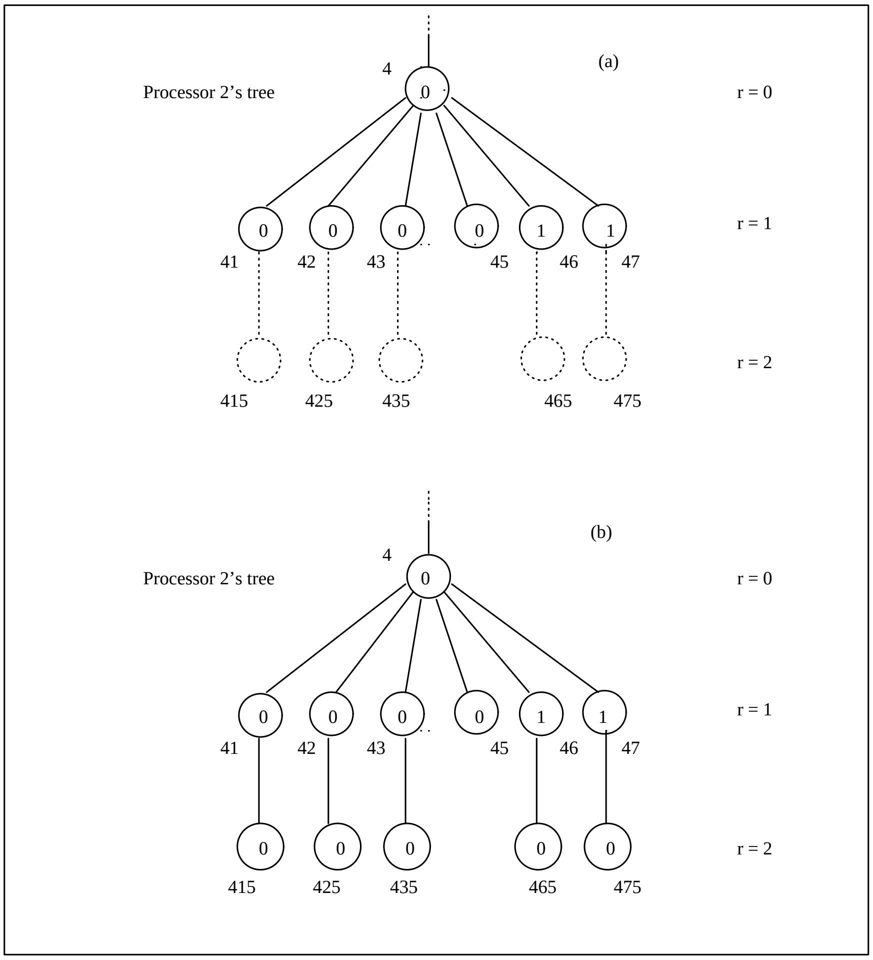

Fig. 2. Safe message reconstruction.

Theorem 12. Safe EIG *solves the Distributed Consensus problem using* $n > 3t$ *processors and* $min(t+1, f+2)$ *communication rounds.*

While the consensus protocol of Theorem 12 is processor- and round-optimal, it is not efficient—its maximum message size is $O(t!)$. In the next section we present protocols that are early-stopping and use messages of constant size.

3 Early Stopping with Constant Message Size

The protocols of this section are based on the *Phase Queen* and *Phase King* protocols of [BG1, BGP]. We first show how to transform them into early-stopping protocols—without jeopardizing their simplicity, and then demonstrate several techniques to further reduce the round and bit (and message) complexity while maintaining the early-stopping property.

The first early-stopping consensus protocol—*Early-Stopping Phase Queen* (*ESPQ* for short)—requires $n > 4t$ and is shown in Figure 3. It executes at most $t+1$ phases each consisting of two communication rounds: the universal exchange and the broadcast of the "phase queen"; each phase has a distinct queen. (In this section the semantics of Send(U) is to send (the contents of) U to all processors—including the sending processor.) We assume that each processor p has local variables V (the subject of the consensus problem), P (indicating whether or not there is a "strong preference" for V) and S (which counts the number of processors with strong preferences), and integer array $C[0..1]$ (the counts of 0's and 1's received). In the universal exchange each processor changes V to what it perceives as the majority value and sets P to true if V is strongly favored. During the queen's broadcast each processor sends the value determined to be the majority along with an indication of the strength of its preference for this value. After counting the number of processors with strong preferences, a processor replaces its own V with the queen's value if there are too few processors with a strong preference; should it find many processors with a strong preference, it halts. Because some correct processors may halt before others, a convention is adopted to account for messages that are not received during a phase: if no message is received from p during the universal exchange, the receiving processor assumes that the missing value is equal to the value of its own V of the previous phase and that the silent processor indicates a strong preference during the queen's broadcast.

```
for k := 1 to t+1 do begin
                (* universal exchange *)
    send(V);
    for j := 0 to 1 do
                C[j] := the number of received j's;
    V := C[1] > 2t;
    P := C[V] ≥ n − t;
                (* queen's broadcast *)
    send(⟨V, P⟩);
    S := the number of received ⟨v, 1⟩'s, for any v;
    if S ≤ t then
                V := v, where ⟨v, w⟩ is the message
                received from k
    elseif S > 2t then
                halt;
end;
```

Fig. 3. Protocol *Early-Stopping Phase Queen*, code for processor p.

For ease of presentation the queen's broadcast uses two-bit messages. The remark below shows how to reduce this broadcast to one bit and thus make *ESPQ* a single-bit protocol.

Theorem 13. Early-Stopping Phase Queen *solves the Distributed Consensus problem using* $min(2(f+2), 2(t+1))$ *exchange rounds and one-bit messages, where* f *is the number of processors that fail during its execution, provided* $n > 4t$.

Proof. As a preliminary step, we show the following property of protocol *ESPQ* (Fig. 3).
Persistence: If for all $p \in P - T$, $V_p = v$ is true at the start of a phase, every processor decides on v and halts at the end of the phase.

To see this, observe that the universal exchange results in every correct processor p having $C[v]_p \geq n - t > 2t$, $C[\bar{v}]_p \leq t$, and P being true. Therefore it computes $V_p = v$ with a strong preference. During the queen's broadcast, it detects that at least $n - t$ processors have strong preferences so it ignores the queen's broadcast and halts.

Now we can address the correctness of the protocol.
Agreement: Since the number of phases $(t+1)$ exceeds the number of faulty processors (t), there is some first phase g with a correct queen. One of the following cases holds at the end of phase g:

1. $S \leq t$ for each $p \in P - T$. Then every correct processor assigns V_g to V during the queen's broadcast of phase g, where $\langle V_g, w \rangle$ is the message sent by g.
2. $S > t$ for some $p \in P - T$. Then P is true for at least one correct processor p' so at the end of the universal exchange, $V_{p'} = v$ and $C[v]_{p'} > n - t$. Among the $n - t$ processors from which p' received value v, at most t are faulty so at least $n - 2t$ are correct. Thus, *every* correct processor received v at least $n - 2t > 2t$ times, and $\bar{v}$ at most $2t$ times. As a consequence, at the end of the universal exchange $V_q = v$ for every $q \in P - T$, including queen g; therefore the queen's broadcast must result in $V_q = v$ for all $q \in P - T$.

Persistence assures that no correct processor changes its value subsequent to phase g.
Validity: Follows trivially from *Persistence*.
As for the running time, *Agreement* shows that, if the queen of phase k is correct, then for all $p \in P - T$, $V_p = v$ for some v. By *Persistence*, all processors that have not already halted will halt by the end of phase $k+1$. If k is the phase with the first correct queen, the number of faulty processors f is at least $k - 1$ (because all prior queens were faulty). Thus $k + 1 \leq f + 2$. □

Remark: If a correct processor p asserts that it has a strong preference during the queen's broadcast of a phase, then $C[v]_p \geq n - t$ at the end of the phase's universal exchange, for some v. Since at most t of the processors in this count are faulty, $C[v]_q \geq 2t$ for for all $q \in P - T$ so $V_q = v$. Therefore, a correct processor with a strong preference need not send the value that it prefers during the queen's broadcast; it may be deduced by the receiving process as being equal to its own V. We can change the code so that a queen sends its own V during the queen's broadcast if and only if it does not have a strong preference for V, and let all other processors interpret a missing message from the queen in this light. Therefore, *ESPQ* may use single-bit messages.

ESPQ is asymptotically optimal with respect to rounds and number of processors but is exactly optimal in message size. The second protocol—*Early-Stopping Phase King* (*ESPK*)—sacrifices exact optimality in message size (it uses two-bit messages) in order to achieve exact optimality in the number of processors, i.e., $n > 3t$. The protocol is shown in Figure 4.

```
for k := 1 to t+1 do begin
            (* universal exchange 1 *)
    send(V);
    for j := 0 to 1 do
            C[j] := the number of received j's;
            (* universal exchange 2 *)
    for j := 0 to 1 do
            send(C[j] ≥ n − t)
            D[j] := the number of received 1's;
    V := D[1] > t;
    P := D[V] ≥ n − t;
            (* king's broadcast *)
    send(⟨V, P⟩);
    S := the number of received ⟨v, 1⟩'s, for any v;
    if S ≤ t then
            V := v, where ⟨v, w⟩ is the message
            received from k
    elseif S > 2t then
            halt;
end;
```

Fig. 4. Protocol *Early Stopping Phase King*, code for processor p.

ESPK is like *ESPQ* with an additional universal exchange. The additional exchange can be either one round using two-bit messages, or two rounds using single-bit messages. After this exchange, we can show that there is at most one v for which $D[v]_p > t$ for any processor p. Therefore, a processor can compute a new value for V using fewer processors (t versus $2t$) than in *ESPQ*.

Theorem 14. Early-Stopping Phase King *solves the Distributed Consensus problem using* $min(3(f+2), 3(t+1))$ *exchange rounds and two-bit messages, or* $min(4(f+2), 4(t+1))$ *exchange rounds using single-bit messages, where* f *is the* *number of processors that fail during its execution, provided $n > 3t$.*

The proof is very similar to that of *ESPQ* (Theorem 13) and is omitted from the abstract.

3.1 Reducing the Number of Rounds

While the two early-stopping consensus protocols presented above have optimal message size, their round complexity is at least twice optimal. Using the committee technique of [BG1], we show in the full paper that the number of rounds may be reduced to $\min(f(1+1/d)+5, t(1+1/d))$ for every constant d while maintaining the same number of processors and constant message size.

Briefly, the technique involves replacing the queen (king) processor of each phase with a *committee* of processors. Intra-committee the processors run any round-optimal early-stopping distributed consensus protocol. Any processor not in the committee of a phase merely listens and uses as the queen's (king's) message any value decided on by at least t processors in the committee. By partitioning the processors into R disjoint committees, the protocol can be run for at most R phases (versus $t+1$). Because of the intra-committee early-stopping, some processors may advance to the next phase before others; the full paper shows that this loss of total synchrony can be overcome without penalty in the round complexity.

If the intra-committee protocol of phase m can tolerate $t_m - 1$ faulty processors, then a *bad committee* (one in which more than $t_m - 1$ processors are faulty) runs for at most t_m rounds whereas a *good comittee* (one with $f_m < t_m$ faulty processors) runs for $min(f_m + 2, t_m + 1)$ rounds. Thus, the total number of rounds in a run in which the first good committee doesn't occur until phase F is:

$$\underbrace{\sum_{m=1}^{F-1} 1 + t_m}_{\text{phases with bad committees}} \quad + \quad \underbrace{1 + (f_F + 2)}_{\text{phase with first good committee}} \quad + \quad \underbrace{2}_{\text{final phase}} \approx f(1 + 1/d) + 5$$

where $f = \sum_{m=1}^{F-1} t_m + f_m$.

4 Early Stopping with Optimal Bit and Message Complexity

In the full paper we show that the committee technique of the previous section may be applied in the extreme (as indicated in [BGP]) to obtain an early-stopping protocol with exact processor optimality, near round optimality, and asymptotically optimal bit (and message) complexity ($O(nt)$). The technique splits the processors into two committees, each acting the role of king in one of two phases of the *Early-Stopping Phase King* protocol. Intra-committee, the processors reach consensus by recursively applying the technique until the committee size becomes too small, in which case they use the *ESDM* protocol of [BG2]. We note that a straightforward application of the technique of [BGP] does not suffice: extra measures must be taken to prevent even a single faulty processor from delaying termination for a number of rounds equal to the depth of the recursion. Note that our protocol achieves message optimality in *all* runs (at the expense of a reasonably small constant), unlike the protocol of Hadzilacos and Halpern [HH] which only achieves exact message optimality for *failure-free* runs.

Acknowledgements

The authors would like to thank Gil Neiger for his help in tracking down the message size of existing early-stopping consensus protocols, and Yoram Moses for commenting on an early draft of the manuscript and improving the presentation.

References

[BDDS] A. Bar-Noy, D. Dolev, C. Dwork and H.R. Strong, "Shifting gears: changing algorithms on the fly to expedite Byzantine Agreement," *Proc. 6th PODC*, pp. 42-51, August 1987.

[BG1] P. Berman and J.A. Garay, "Asymptotically Optimal Distributed Consensus," *Proc. ICALP 89*, LNCS Vol. 372, pp. 80-94, July 1989.

[BG2] P. Berman and J.A. Garay, "Distributed Consensus with $n = (3 + \varepsilon)t$ Processors," *Proc. 5th International Workshop on Dist. Algorithms*, LNCS, Springer-Verlag, October 1991.

[BGP] P. Berman, J.A. Garay and K.J. Perry, "Towards Optimal Distributed Consensus," *Proc. 30th FOCS*, pp. 410-415, October/November 1989.

[C] B. Coan, "Efficient agreement using fault diagnosis," *Proc. 26th Allerton Conf. on Comm., Control and Computing*, pp. 663-672, 1988.

[DR] D. Dolev and R. Reischuk, "Bounds of Information Exchange for Byzantine Agreement," *JACM*, Vol. 32, No. 1, pp. 191-204, 1985.

[DRS] D. Dolev, R. Reischuk and H.R. Strong, "Eventual is Earlier than Immediate," in *Proc. 23rd STOC*, 1982. Revised version appears in "Early Stopping in Byzantine Agreement," *JACM*, Vol. 37, No. 4 (1990), pp. 720-741.

[DS] D. Dolev and H.R. Strong, "Authenticated Algorithms for Byzantine Agreement," in *SIAM Journal of Computing*, Vol. 12, pp. 656-666, 1983.

[FM] P. Feldman and S. Micali, "Optimal Algorithms for Byzantine Agreement", *Proc. 20th STOC*, pp. 148-161, May 1988.

[HH] V. Hadzilacos and J. Halpern, "Message-Optimal Protocols for Byzantine Agreement," *Proc. 10th PODC*, pp. 309-324, August 1991.

[HM] J. Halpern and Y. Moses, "Knowledge and common knowledge in a distributed environment," *JACM*, Vol. 37, No. 3 (1990), pp. 549-587.

[LF] L. Lamport and M. Fischer, *Byzantine Generals and Transaction Commit Protocols,* Opus 62, SRI International, April 1982.

[LSP] L. Lamport, R.E. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM ToPLaS*, Vol. 4, No. 3, pp. 382-401, July 1982.

[MW] Y. Moses and O. Waarts, "Coordinated Traversal: (t+1)-Round Byzantine Agreement in Polynomial Time," *Proc. 29th FOCS*, pp. 246-255, October 1988.

[PT] K.J. Perry and S. Toueg, "Distributed agreement in the presence of processor and communication faults," *IEEE Trans. on Software Engineering*, Vol. 12, No. 3, pp. 477-482, March 1986.

[TPS] S. Toueg, K.J. Perry and T.K. Srikanth, "Fast Distributed Agreement," *SIAM Journal of Computing*, Vol. 16, No. 3, pp. 445-457, June 1987.

[W] O. Waarts, "Coordinated Traversal: Byzantine Agreement in polynomial time," M.Sc. Thesis, Weizmann Institute of Science, Rehovot, Israel, August 1988.

[Z] A. Zamsky, "New Algorithms for Agreement Problem in Synchronous Distributed Networks," M.Sc. Thesis, Technion, Haifa, Israel, April 1992.