Cloture Votes: n/4-resilient Distributed Consensus in t+1 rounds†

Piotr Berman
Department of Computer Science
The Pennsylvania State University
University Park, PA 16802
berman@theory.cs.psu.edu

Juan A. Garay
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
garay@watson.ibm.com

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. We measure the quality of a consensus protocol using the following parameters: total number of processors n, number of rounds of message exchange r, and maximal message size m. The known lower bounds are respectively $3t+1$, $t+1$ and 1.

While no known protocol is optimal in all these three aspects simultaneously, *Cloture Votes*—the protocol presented in this paper—takes further steps in this direction, by making consensus possible with $n = 4t+1$, $r = t+1$ and polynomial message size.

Cloture is a parliamentary procedure (also known as "parliamentary guillotine") which makes it possible to curtail unnecessary long debates. In our protocol, the unanimous will of the correct processors (akin to parliamentarian supermajority) may curtail the debate. This is facilitated by having the processors open in each round a new process (debate), which either ends quickly, with the conclusion "continue" or "terminate with the default value," or lasts through many rounds. Importantly, in the latter case the messages being sent are short.

---

† Preliminary version appeared as a part of "Towards Optimal Distributed Consensus" in *Proc. 30th IEEE Symp. on Foundations of Computer Science* [BGP], and is part of the second author's Ph.D. dissertation.

1. Introduction

The *Distributed Consensus* problem is one of the central topics in the theory of reliable distributed systems. Given a set P of processors, we are told that processors in some unknown subset T of size at most t are *faulty*. The goal of the problem is to cause the *correct* processors (i.e., those in $P-T$) to agree on an outcome regardless of the behavior of the faulty processors.

The standard model used in studying this problem is of a network of n synchronous processors numbered from 1 to n. The computation performed by the network evolves as a series of *rounds*. In each round, the correct processors exchange messages over reliable links and perform local computations according to the protocol. In contrast, the faulty processors may exhibit arbitrary (*Byzantine*) behavior, even appearing to act maliciously in concert. Each processor is given an initial value, 0 or 1. A protocol solves the Distributed Consensus problem if it always terminates and the correct processor's final values satisfy the following two conditions:

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

This is a variant of the *Byzantine Agreement* problem in which the processors are required to agree on a value sent to them by one of the processors (called the *general*) in an initial round [LSP].

There are three basic aspects contributing to the quality of a protocol for Distributed Consensus: the resiliency—which relates the total number n of processors and the maximum tolerable number t of faulty processors, the number of rounds r, and the cost of communication, given by the maximum message size m (while in this paper we omit the explicit discussion of the computational complexity, in the protocols presented here the latter is always determined by their message size). The optima for these quality parameters are respectively $n=3t+1$ [PSL], $r=t+1$ [DS, FL] and $m=1$ (obvious). These bounds are independent in the sense that they are valid no matter what the values of the other parameters are.

Motivated by practical concerns, a recent focus of research has been on the efficiency of protocols that solve the problem (e.g., [BD, BDDS, BG1, CW1, FM, MW]), although no protocol as yet is optimal in all three aspects simultaneously. In contrast, in the probabilistic setting Feldman and Micali [FM] have achieved optimal resiliency, constant expected number of rounds and polynomial message size. Figure 1 shows the existing deterministic consensus protocols that run in the optimal number of rounds, the focus of this paper. The original protocol of Lamport, Shostak and Pease (and its refinement, due to Bar-Noy *et al.* [BDDS]) was resiliency- and round-optimal, but required messages of exponential size. The protocols of Bar-Noy and Dolev [BD], and Coan [C2], achieved message size optimality, but at the expense of quadratic number of processors. The first protocol with optimal number of rounds, linear resiliency ($n > 6t$) and polynomial message size was provided by Moses and Waarts [MW]. This result was unexpected, and contradicted a previous conjecture [BD]. In this paper we present an efficient consensus protocol—*Cloture Votes*—which further narrows the optimality gap by achieving consensus with $n > 4t$ while being significantly simpler than the Moses and Waarts protocol, at the expense of a moderate increase in message size.†

	n	m
Lower bounds [PSL]	$3t+1$	1
Lamport *et al.* [LSP], Bar-Noy *et al.* [BDDS]	$3t+1$	$O(n^t)$
Bar-Noy and Dolev [BD], Coan [C2]	$\Omega(t^2)$	1
Moses and Waarts [MW]	$6t+1$	$O(n^4)$
Cloture Votes	$4t+1$	$O(n^5)$

Fig. 1. Distributed Consensus in optimal number of rounds ($r = t+1$ [DS]).

Our method borrows from the parliamentary procedure known as *Cloture*: weary of long speeches (messages), the senators (processors) may vote for termination; in this way a qualified majority of the senators may swiftly curtail the debate. Similarly as the Moses and Waarts protocol, *Cloture Votes* is used in conjunction with the two earlier techniques of *fault masking* [C1] and *early stopping* [DRS]. In particular, its development was influenced by the *gear shifting* technique of Bar-Noy *et al.* [BDDS].

Our solution to Distributed Consensus provides equally efficient solutions to the related *Byzantine Agreement* [LSP] and *Firing Squad* [BL] problems. The *Cloture Votes* protocol, together with the schema used by the authors to reduce the number of rounds of a consensus protocol [BG1], yields a family of protocols that achieve consensus with $n > 4t$, $r = t + t/d$ and $m = O(n^d)$, for every constant d. This tradeoff compares favorably with the exponential (in d) message size in similar protocol

† In the final section we indicate how extensions of the technique can be used to reduce the message size of *Cloture Votes* to $O(n^4)$.

families of Coan [C1] and Bar-Noy *et al.* [BDDS].

The remainder of the paper is organized as follows. In Section 2 we introduce the *Cloture Votes* protocol by applying successive transformations to the version of the protocol of Lamport *et al.* [LSP] due to Bar-Noy *et al.* [BDDS]. In Section 3 we show how to modify the protocol to satisfy the *early stopping* property [DRS]; this allows *Cloture Votes* to stop in $r = \min\{t+1, f+2\}$, where f is the actual number of faults that occur during the execution of the protocol. We conclude with some final remarks.

2. Cloture Votes: t+1-round, Polynomial Time Consensus for n > 4t

2.1. Exponential Information Gathering

In this subsection we describe the formulation of the consensus protocol of [LSP] by Bar-Noy *et al.* [BDDS], called *Exponential Information Gathering with Recursive Majority Voting* by the authors (*EIG* for short). We reformulate the correctness proof in a way more suitable to explain the transformations that will reduce the message size of the final protocol (*Cloture Votes*) to a polynomial, while maintaining round optimality for $n > 4t$. While *EIG* takes the optimal number of rounds to terminate, we additionally show how the running time can be reduced under some favorable conditions; this result will also be used by *Cloture Votes*.

In *EIG* processors build trees of the following form: the set of nodes consists of the sequences from P^* without repetitions of length at most $t+1$; the root is λ (the empty sequence); the children of node σ, $EXT(\sigma)$, are nodes of the form σj. (Note that the depth of node σ, as defined, equals $|\sigma|$, the length of the corresponding sequence.) In each tree, a node σ has a *value* denoted by $Val(\sigma)$. $Val(\sigma)$ is assigned twice during the execution of the protocol: a preliminary value is assigned during the initialization and the $t+1$ message exchange rounds, and a final value is assigned with the use of the majority operator during the final computation stage. During the first part of the protocol the values of nodes in processor i's tree have the following intuitive meaning: $Val(j)$ stores the initial value of j as reported to i by j, $Val(jk)$ stores the value that j reported to k, as reported to i by k, etc. We say that node σ is *v-common* if v is σ's final value in the trees of all correct processors; if the particular value of v is irrelevant, we just say that σ is *common*.

In the protocols of this paper processors communicate using the following two primitives:

- **Send**(σ): send $Val(\sigma)$ to all processors (including the sender itself).
- **Receive**(σ, j): upon receiving v when j (presumably) executes Send(σ), create node σj and set $Val(\sigma j) = v$. (Here processors also receive the values sent by themselves.)

The *EIG* protocol is shown in Figure 2; *CL* is the acronym for "Current Level." The majority operator is defined in such a way that in case of a tie, processors choose 0, the default value. We now prove the correctness of *EIG*, assuming $n > 3t$.

```
create node λ; Val(λ) := Initial;
CL := {λ};

    (* communication of the preliminary values *)
for r := 0 to t do begin (* execute t+1 rounds *)
    for σ ∈ CL do
        Send(σ);
    for σ ∈ CL do
        for j ∈ Ext(σ) do
            Receive(σ,j);
    CL := ⋃_{σ∈CL} EXT(σ)
end;

    (* computation of the final values *)
for r := t downto 1 do begin
    CL := parents of nodes in CL;
    for σ ∈ CL do
        Val(σ) := the majority value of Val(EXT(σ))
end;
write(Val(λ));
```

Fig. 2. Protocol *EIG*: code for processor i.

Lemma 2.1: If the majority of children of σ are v-common, σ is also v-common.

The proof follows directly from computing the final values via majority.

Lemma 2.2: Assume $|\sigma| \le t$, $\sigma i \in EXT(\sigma)$ and i is a correct processor. Then v is the preliminary $Val(\sigma)$ for i if and only if σi is v-common.

Proof: By decreasing induction on $|\sigma|$. For $|\sigma| = t$, in round t processor i sends v via Send(σ), and all correct processors perform $Val(\sigma i) := v$ in Receive(σ, i); later this value is not modified.

For $|\sigma| < t$, when i sends v via Send(σ), all correct processors assign v to $Val(\sigma i)$ in Receive(σ, i). Inductively, for every correct $j \in Ext(\sigma i)$, the node $\sigma i j$ is

v-common. Since the number of such j's is at least $n-2t>t$, Lemma 2.1 applies. □

Corollary 2.3: If i is a correct processor, then all nodes of the form σi are common.

The following lemma concerns nodes (λ included) whose sequences consist of faulty processors only.

Lemma 2.4: All nodes in T^* are common.

Proof: Assume otherwise, and let σ be the node of largest depth such that neither σ, nor any of its prefixes (ancestors) are common. Corollary 2.3 implies that $\sigma \in T^*$, thus $|\sigma| \le t$ (since t is the upper bound on the number of faults, and no repetitions occur in σ), and so σ has children. Since all the children of σ are common, the majority of them are v-common, for some v. Hence, by Lemma 2.1 σ is v-common as well, a contradiction. □

Theorem 2.1: Protocol *EIG* solves the Distributed Consensus problem, provided $n>3t$.

Proof: The Agreement condition follows from Lemma 2.4. Concerning Validity, if $Initial = v$ for all correct processors, then v is the preliminary value of λ in their trees. Hence, by Lemma 2.2, node i is v-common in all processors' trees, for every $i \in P-T$, and so is, by Lemma 2.1, the root λ. □

While *EIG* uses the optimal number of rounds, later in our *Cloture Votes* protocol we will invoke a consensus protocol in every round; therefore, these protocols have to run in a number of rounds smaller than $t+1$. Favorable conditions in which we invoke these protocols will allow us to guarantee consensus in these cases. We now formally specify these conditions.

We say that a run of a protocol satisfies the *unanimity* condition if all the correct processors start off with the same value of *Initial*. We also say that the run is s-*favorable* if at least s ($s \le t$) of the faulty processors send value 0 in all their messages.

Let protocol EIG_s be the same as *EIG* of Figure 2, except for replacing the fourth line by:

for r := s **to** t **do begin** (* execute $t+1-s$ rounds *)

The following theorem shows that EIG_s suffices to solve the Distributed Consensus problem if we can assure that one of the above conditions hold. We will use Z to denote the subset of the faulty processors that send 0 in all their messages. For convenience, we assume that processors from Z assign 0 to all their $Val(\sigma)$ variables, and then execute their send operations correctly.

Theorem 2.2: Protocol EIG_s solves the Distributed Consensus problem in those runs that either satisfy the unanimity condition or are s-favorable, provided $n > 3t$.

Proof: Validity is assured because Lemmas 2.1 and 2.2 apply to EIG_s. Thus, we only need to prove that Agreement is satisfied, assuming the run is s-favorable.

First, observe that Lemma 2.2 and Corollary 2.3 apply to processors from Z as well. This allows us to show that all the nodes from $T - Z$ are common in all the correct processors' trees, following the proof of Lemma 2.4. Hence, λ is common in the processors' trees, and Agreement follows. □

Note that in EIG_s the maximum message size is $m = O(n!/(n-t+s)!)$. In the next subsection we describe two techniques to "prune" some of the branches of the EIG trees, thus reducing the size of messages.

2.2. Protocol Debate: EIG with Early Stopping and Fault Masking

>From now on we assume $n > 4t$. The protocol we describe in this subsection (*Debate*) allows correct processors to reach the same conclusions as *EIG* does, but with a smaller amount of communication. This happens because of the following two reasons:

1) For many nodes the final values may be predicted before the end of the communication phase of protocol *EIG* (see Fig. 2), thus making the communication of many values unnecessary. Processors make such predictions through the application of the *early stopping* rules (c1, c2, c3) described below; as a result, they may stop early sending messages concerning the descendants of such nodes.

2) Once a processor discovers that another processor, say j, is faulty (by applying rule (fd) below), it may pretend that all values reported to it by j are 0, without jeopardizing the correctness of the protocol (*fault masking* [C]).

Fault masking is crucial not only because it increases the cases where the early stopping rules apply, but also—as we will see in the next subsection—because it makes possible to assure that s-favorable conditions hold for instances of the *Debate* protocol, whenever *Unanimity* is not satisfied. In particular, a *detected* faulty processor—i.e., a processor discovered as faulty by all correct processors—effectively belongs to Z.

To discuss the inferences made by the processors concerning either predictions of final values for some nodes or discoveries of faults, we will use notation from the theory of knowledge in distributed systems [HM]. We read $K_i^r\phi$ as "after round r 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. Instead of describing data structures, we will provide computationally easy rules that make it possible to deduce statements of the form $K\phi$. We note that only certain specific ways of inferring knowledge will be allowed. Therefore a statement of the form $\neg K_i^r \phi$ will mean that the rules that we provide do not allow i to deduce ϕ in round r. We also assume that processors do not "forget," i.e. $K_i^r \phi$ implies $K_i^{r+1} \phi$.

We now provide the rules used by the correct processors in protocol *Debate* to deduce facts of the form [σ is v-common]. These rules are valid provided $\sigma \in (T-Z)^* \cup (T-Z)^*P$. This is sufficient for our purposes, because we will prove that in *Debate* processors do not apply these rules to any other nodes.

K_i^r[σ is v-common] if one of the following holds in i's tree at the end of round r:

(c1) more than $n/2+t-|\sigma|$ children of σ have value v (as σ has at most $t-|\sigma|$ faulty children);

(c2) K_i^r[τ is v-common] for a majority of children τ of σ;

(c3) v is the value of σ and node σ is created in round $r=t$ (as $\sigma \in T^*P$).

Below we show important features of the knowledge operators.

Lemma 2.5: Assume that node $\sigma \in (T-Z)^*(P-(T-Z))$ is created before round r. Then E^r[σ is v-common], for some v.

Proof: Node σ is of the form τi, where i is either correct, or is a member of Z. Assume that in i's tree the preliminary $Val(\tau)$ equals v. Because i executes Send(τ) correctly, all correct processors assign v to $Val(\tau i)$ during the execution of Receive(τ,i). If σ is created in round t, the correct processors can apply rule (c3). Otherwise, in the next round processors execute Send(σi), and as a result, each of them creates in its tree at least $n-t-1 > n/2+t-|\sigma|$ children of σ with preliminary value v. Thus, rule (c1) implies E^r[σ is v-common]. □

Lemma 2.6: Assume that the unanimity condition holds for EIG_s. Then E^s[λ is common].

Proof: Since in the tree of each correct processor $n-t > n/2-t$ children of λ store the same value, the claim follows from rule (c1). □

As we will see, once a processor knows that node σ is v-common, it does not need to establish final values for the descendants of σ. Therefore, the question of the form "is σ v-common" will never arise for nodes σ not in $(T-Z)^* \cup (T-Z)^*P$.

Lemma 2.7: Let $i \in P-T$ and $\sigma \in (T-Z)^* \cup (T-Z)^*P$. Then $K_i^r[\sigma$ is v-common] implies $E^{r+1}[\sigma$ is v-common].

Proof: For $\sigma \in (T-Z)^*(P-(T-Z))$, $K_i^r[\sigma$ is v-common] implies $E^r[\sigma$ is v-common] by Lemma 2.5.

Consider now $\sigma \in (T-Z)^*$, created in round $r-1$. In this case i had to deduce that σ is v-common by applying rule (c1). If $r = t$, then all the children of σ are common and the claim holds. Otherwise $(r<t)$, note that in i's tree more than $n/2+t-|\sigma|$ children of σ have preliminary value v, and more than $n/2$ among them are of the form σj, where j is correct. By Lemma 2.5 for such children we have $E^{r+1}[\sigma j$ is v-common], and thus $E^{r+1}[\sigma$ is v-common] by rule (c2).

For σ created before round $r-1$, the claim follows by induction and application of rule (c2). The base case of the induction is provided above, and the (decreasing) parameter of the induction is the round number in which σ is created. (Alternatively, the increasing parameter of induction is the depth of the proof that allows processor i to deduce that σ is v-common.) □

Lemma 2.6 shows that once correct processor i knows that all processors will establish v as the final value of σ, all other processors will know it after receiving the next round's messages. Thus, after sending messages in the next round concerning the descendants of σ, i may stop receiving and sending any such messages. This is exactly the method of *early stopping* [DRS].

As a consequence, if in a given round r of the protocol correct processor i stores a value for node σ, and yet it needs to read messages for the descendants of σ, then $\neg K_i^r[\tau$ is common], for every ancestor τ of σ. We say in this case that σ is *undecided* in round r. The *fault masking* technique [C1] exploits the fact that if node σj is undecided in round r, then every processor in the sequence σ is faulty. More formally, we let the correct processors use the following rule for the discovery of faults:

(fd) If $\#\{k : Val(\sigma jk) \neq Val(\sigma j)$ or $K_i^r[k$ is faulty$]\} > t$, then $K_i^r[j$ is faulty].

Fault masking means that if $K_i^r[j$ is faulty] then, starting in round r, processor i replaces all the messages from j by 0 (the default value). This does not disturb the correctness of the protocol, because *EIG* works regardless of the behavior of the faulty processors. In the next subsection we will see (Lemma 2.10) how fault masking limits the ability of the faulty processors to create undecided nodes.

Lemma 2.8: Assume that a correct processor i creates node σ before round r, and σ is undecided in round r. Then, for every j in the sequence σ, $K_i^r[j$ is faulty] and $j \notin Z$.

Proof: Let τj be the shortest sequence violating the claim. Because τ is undecided as well, we have $K_i^r[k$ is faulty] for every k in the sequence τ. Because $\neg K_i^r[j$ is faulty], out of the $n-|\tau j|$ children of τj in σ's subtree, at most $t-|\sigma|$ of them have a value different from $Val(\tau j)=v$. Consequently, at least $n-t-1>n/2+t-|\tau j|$ of these children have value v and, by rule (c1), $K_i^r[\tau j$ is v-common]. But then τj is not undecided, a contradiction.

Lastly, note that if $j \in Z$, rule (fd) cannot be used to deduce that j is faulty. □

We call $Debate_s$ the protocol obtained from EIG_s by incorporating the two modifications—early stopping and fault masking—described above. The protocol is shown in Figure 3, and is run by every correct processor with parameter $Start = s$. The consensus value is stored as $Outcome[s]$ (later, *Debate* is repeatedly invoked by protocol *Cloture Votes*). In the protocol, every processor maintains a tree; *Val* maps nodes of *Tree* into $\{0,1\}$. The set U consists of the nodes from the current level of the tree that are undecided, as defined above (i.e., nodes for which children need to be created in order to establish their final values). The set M consists of nodes which will be the subject of messages sent to the other processors (recall that messages are created by Send(σ) operations, where σ is a node). Because sets U and M are used instead of their superset *CL* (Current Level) of *EIG*, *Debate* has much shorter messages.

As in *EIG* (Fig. 2), processors send the nodes' preliminary values, and use the majority operator to compute the final values. Because processors use rules (c1,c2,c3), many of the final values in the tree can be predicted before the end of the $t+1$ communication rounds. Also, because processors apply fault discovery rule (fd), they are able to diagnose some of the faulty processors during the computation, and fault masking alters the messages received from the discovered faulty processors in a way which increases the applicability of the early stopping rules (c1,c2,c3). Recall that the goal is to compute the final value of the root, thus, once a processor computes a node's final value, it does not need messages concerning this node's descendants any longer, and they can be removed from set U. The descendants of the removed nodes are kept for one more round in M, because other processors may need to know their values (Lemma 2.7 shows that this one extra round is sufficient). As a consequence, if U becomes empty at the end of round r in processor i's tree, M will become empty in round $r+1$, and processor i will stop (early) communicating with the other processors. Thus, we will regard setting M to $\emptyset$ as the termination of this protocol. (This aspect of the protocol will be relevant when we show in Section 3 how to terminate the execution of protocol *Cloture Votes* in a number of rounds proportional to the actual

```
procedure Debate(Start, Initial);
begin
    create node λ;  Val(λ) := Initial;
    Tree, M, U := {λ};
    for r := Start to t do begin (* execute t+1−Start rounds *)
        for σ ∈ M do
            Send(σ);
        M := ⋃_{σ∈U} EXT(σ);
        Tree := Tree ∪ M;
        for every σj ∈ M do
            Receive(σ,j);

                (* fault masking *)
        for every τj ∈ M such that K[j is faulty] do
            Val(τj) := 0;

                (* early stopping *)
        for every σ ∈ Tree and v such that K[σ is v-common] do
            begin
                Val(σ) := v;
                remove the descendants of σ from U
            end;
        U := ⋃_{σ∈U} EXT(σ)
    end;
    Outcome[Start] := Val(λ)
end;
```

Fig. 3. Protocol *Debate*: code for processor i.

number of occurring faults.)

We now provide the counterpart of Theorems 2.1 and 2.2, for $n > 4t$:

Theorem 2.3: Protocol *Debate*(*Start*, $\cdot$) solves the Distributed Consensus problem in those runs that either satisfy the unanimity condition or are *Start*-favorable, provided $n > 4t$.

Proof: Validity follows from Lemma 2.5, and Agreement from Lemma 2.6. Essentially, *Debate*(*Start*, $\cdot$) sends all the messages of EIG_{Start} that are needed by the correct processors to deduce that λ is v-common, for the consensus value v. □

Although more efficient than EIG_{Start}, the communication cost of protocol *Debate*(*Start*, $\cdot$) is still exponential. In fact, one can apply the results of the next subsection (Lemma 2.10) to bound the maximum message size by $O(2^t n^3)$; we leave the details to the reader. In the next subsection, we show how to further reduce the message size to a polynomial, while maintaining round optimality for $n > 4t$ processors.

2.3. The Cloture Votes Protocol

In *Cloture Votes* each correct processor runs several processes concurrently, together with an overhead computation (the *MAIN* process). A new process is started in each of the $t+1$ rounds; a process started in round s, $0 \le s \le t$, is an execution of procedure $Debate_s$ from the previous subsection. If processor i starts $Debate_k$ with value v (i.e., $Debate(k,v)$), we will sometimes say that i *votes* for v in round k. We assume that in every round there is a single time across processes at which messages are sent and, next, a single time during which messages are received.

The reason for having parallel debates is the following: $Debate_0$ should reconcile the original vote; when the run satisfies the unanimity condition, this debate and all subsequent ones terminate right away. Otherwise, the message size in the original debate may grow. However, if it does grow then clearly the default can be safely chosen. Therefore, once a processor is required by the algorithm to send a very big message (not unlike a filibuster speech in the American Senate), it will cast a vote for cloture. Thus, in every round a new debate starts—shall the deliberation end? (this is exactly what a parliamentary cloture vote is about). Although the cloture vote itself may be debated, once "filibuster speeches" exceed certain polynomial size, the cloture vote will be unanimous and the deliberation will actually end. Moreover, if even one of the correct processors casts a vote for cloture in $Debate_k$, the protocol assures a k-favorable condition holds for this particular debate. Thus, all the debates conclude successfully (i.e., agreement is reached in all of them). A side effect of this setup is an asymmetric early stopping property: an early agreement on the default value in any of the debates halts the protocol. On the other hand, even after achieving early agreement on the non-default value in $Debate_0$, new debates are still started. Section 3 explains how to modify this scheme to assure early stopping in all cases.

A high-level sketch of *MAIN* is given in Fig. 4; inter-process synchronization and communication details are left implicit. We note that the scope of facts explicitly involving nodes is local, i.e. restricted to the corresponding debates, whereas facts of the form [processor i is faulty] are globally known, and shared by all the debates. Parameter S, the *patience* threshold, is set to n^3; justification for this particular value is given in Lemma 2.10.

Lemma 2.9: (*Validity*) Assume that all processors start *Cloture Votes* with the same input value v. Then all correct processors output v.

Proof: Under the assumption, in $Debate_0$ we have $E^0[\lambda$ is v-common]. As a result, after computing $M = P$ in round 0, all correct processors compute $Val(\lambda) = v$ and $U =$

```
Round 0 of MAIN:
    S := n^3;
    start Debate(0,Initial);

Round k, 1 ≤ k ≤ t:
        (* cloture *)
    if K[λ is 0-common] for some Debate_l then begin
        send messages corresponding to Debate_l only
        (choose arbitrarily, if many such l's);
        write(0);
        halt
    end;
        (* start this round's debate *)
    if for an ongoing debate #M ≥ S then
        start Debate(k,0)  (* vote for cloture *)
    else
        start Debate(k,1);

At the end of round t:
    write( MIN_{0≤k≤t} Outcome[k]).
```

Fig. 4. The *Cloture Votes* protocol: code for processor i.

$\varnothing$ (see Fig. 3).

In round 1 no correct processor reads $Debate_0$ messages, and all of them start $Debate(1,1)$: the condition to vote for cloture is not satisfied, as they have $\#M = n$ (see Fig. 4). Moreover, in $Debate_0$ they set $M = \varnothing$. If $v = 0$, this is the last round, and even if $v = 1$, in the subsequent rounds correct processors do not do anything in $Debate_0$.

If $v = 1$ the situation repeats itself: at the beginning of round r correct processors have empty U in the first r debates, empty M in the first $r-1$ debates and $M = P$ in $Debate_{r-1}$. Thus all of them start $Debate(r,1)$, and set $M = \varnothing$ in $Debate_{r-1}$. At the end, all correct processors have $Outcome[0], Outcome[1], Outcome[2], \cdots$ equal to $1,1,1, \cdots$ Hence, all of them write $MIN(1,1,...) = 1$. $\Box$

The next lemma provides the crucial reason that makes *Cloture Votes* work. When the messages corresponding to a debate become long, the correct processors know (Lemma 2.9) that any outcome is permissible. This gives them a motivation to cut the protocol short and opt for the default value. However, any attempt of this kind is essentially equivalent to the original consensus problem, and there is a danger that the remaining number of rounds could be insufficient to assure such a consensus. As it follows from [DS], every faulty processor which is undetected (i.e., $j \in T$ for which $\neg E^r[j$ is faulty$]$) may delay the consensus by one round. On the other hand, detected faulty processors have all their messages masked, and so they virtually belong to Z, the set of processors that send 0's only. Thus, correct processors may risk a non-unanimous vote in round r if some r faulty processors are detected, and consequently the run is r-favorable at that point.

We will show that this indeed is the case when the message size exceeds the patience threshold S. Let $F^r = \{j : E^r[j$ is faulty$]\}$, i.e., the set of processors detected by the end of round r. Note that due to fault masking, an r-favorable condition for $Debate_r$ is assured when $\#F^r \geq r$. We first prove the following technical lemma.

Lemma 2.10: (*Sufficient Fault Detection*) If $\#M \geq 2^m n^3$ holds in $Debate_k$ for a correct processor at the beginning of round s, then $\#(F^{s-3} - F^k) \geq s+m-k$.

Proof: If a given node, say σ, is created in round $r-1$, and yet in round r σ is undecided in the trees of all the correct processors, we say that σ is *corrupted*.

By Lemma 2.8, if a node of the form σj created in round $r-1$ is corrupted, then $E^r[j$ is faulty$]$, and hence $j \in F^r$. Moreover, if $j \in F^r$ and a node of the form τj is created in round $r' \geq r$, then all the correct processors set $Val(\tau j) = 0$, and consequently $E^{r'+1}[\tau j$ is common$]$, and τj is not corrupted. From this we can conclude that if a node of the form $j_{k+1} \cdots j_{k+l}$ is corrupted in $Debate_k$, then $j_r \in F^r - F^{r-1}$, for $k < r \leq k+l$.

On the other hand, if at the beginning of round s of $Debate_k$, node $j_{k+1} \cdots j_s \in M$ of a correct processor p, its ancestor $j_{k+1} \cdots j_{s-3}$ is corrupted. Otherwise, for some ancestor τ of the latter node, the following holds: τ is created before round $s-3$; for some correct processor q we have $K_q^{s-3}[\tau$ is common$]$ (because τ is not corrupted), thus (Lemma 2.7) $K_p^{s-2}[\tau$ is common$]$; hence p removes descendants of τ from U in round $s-2$, and descendants of τ cannot be in M after round $s-1$, a contradiction. Similarly, we conclude that $K_p^s[j_{s-2}$ is faulty$]$.

Under our assumptions, we can show that the number of corrupted nodes of the form $j_{k+1} \cdots j_{s-3}$ is less than 2^{m+2} (because $4t < n$). Let $a_r = \#(F^r - F^{r-1})$. By our previous observation, this number is bounded by $\prod_{r=k+1}^{s-3} a_r$. Summarizing, we know that the product of a_r's is more than 2^{m+2}, and that the number of terms in this product is $s-k-3$. Thus, the lemma follows from the following arithmetic

Claim: Assume that $a_1, \cdots, a_m$, s are natural numbers and that $\sum_{r=1}^{m} a_r = m+s$. Then

$\prod_{r=1}^{m} a_r \le 2^s$.

Proof: Without loss of generality, we may assume that no a_r is equal to 0. Then $b_r = a_r - 1$ is also a natural number and the claim reformulates to

$$\prod_{r=1}^{m} (1+b_r) \le 2^{\sum_{r=1}^{m} b_r} = \prod_{r=1}^{m} 2^{b_r}.$$

The latter follows from $1+b \le 2^b$ being true for every natural number b. $\square_{\textbf{Claim}}$ $\square$

Lemma 2.11: (*Agreement*) All correct processors output the same value.

Proof: First observe that we can use Lemma 2.10 to prove by induction that if a correct processor votes for cloture in round k (i.e., it starts $Debate(k,0)$—see Fig. 4), then $\#F^{k-3} \ge s$. Thus, $Debate_k$ starts at a point which is either k-favorable, or satisfies the unanimity condition. Consequently, by Theorem 2.3, all the debates reach consensus, unless the protocol is interrupted by reaching consensus early on value 0 in one of the debates. In either case, the correct processors compute the same final value. $\square$

Lemma 2.12: (*Message Size*) The maximal message size of *Cloture Votes* is $m = O(n^5)$.

Proof: Assume that at the beginning of round r, a correct processor i has $\#M > n^4$ in (some) $Debate_k$. Then for every $\sigma j \in M$ processor i executed Receive(σ,j) in round $r-1$, and therefore for no ancestor τ of σ did i know at the end of round $r-2$ the final value of τ. By Lemma 2.7, no correct processor knew the final value for an ancestor of σ at the end of round $r-3$; even if one of these processors computed such a value in round $r-2$, it would still execute Send(σ) in round $r-1$. Thus, for every correct processor, σ was a member of M in $Debate_k$ at the beginning of round $r-1$. Consequently, for all correct processors, $\#M > n^3$ at the beginning of round $r-1$, and all of them started $Debate(r-1,0)$. This means that the cloture vote of round $r-1$ is successful, i.e., $E^r[\lambda$ is 0-common] in $Debate_r$. Consequently, the correct processors send in round r messages of size $O(n)$ (see "cloture" in Fig. 4).

In the remaining case, a correct processor has $|M| \le n^4$ in all current debates. Moreover, it is possible to code a compound debate message in such a way that its length equals the size of M in the current round plus the size of M in the previous round: the first part of the message codes the current set M in terms of the previous one, the second concatenates bits of individual messages generated by Send's. Thus, the message size in this case is at most $2n^4 t$. $\square$

Lemmas 2.9, 2.11 and 2.12 allow us to prove our main result:

Theorem 2.4: The *Cloture Votes* protocol solves the Distributed Consensus problem in $t+1$ rounds using messages of size $O(n^5)$, provided $n>4t$.

As presented, *Cloture Votes* always terminates in $r=t+1$ rounds. In the next section we show how to modify it in order to stop in a number of rounds that is proportional to the number of faults that actually occur during the execution of the protocol.

3. Early Stopping Cloture Votes

In this section we show how to achieve consensus in $r=\min\{t+1,f+2\}$, where f ($f<t$) is the actual number of occurring faults, and same remaining parameters as before. Dolev *et al.* [DRS] termed this property of a consensus protocol "early stopping," and proved $\min\{t+1,f+2\}$ to be a lower bound for the number of rounds.

Let F_i^r = $\{j: K_i^r[j$ is faulty$]\}$, i.e., the set of processors known to be faulty by processor i at round r. (Compare with F^r, the set of globally detected faults from the previous section.) In this section we assume that *Cloture Votes* runs with patience threshold $S=2n^3$.

By Lemma 2.10, correct processors can start $Debate(s,0)$ only if $\#(F^{s-3}-F^k)>s-k$, for some $k<s$. By induction, this requires $\#F^{s-3}>s$. This allows us to prove the following technical lemma.

Lemma 3.1: (*Fault Deficit*) Suppose that for some $i \in P-T$ and $r>0$, $\#F_i^r<r$. Then either

a) for some $Debate_k$, $k \le r$, $K_i^r[\lambda$ is 0-common], or

b) for every $Debate_k$, $k \le r$, $K_i^r[\lambda$ is 1-common], and for every $l>r$, all correct processors will start $Debate(l,1)$.

Proof: Assume that a) does not hold. To prove b), we first consider $Debate_k$, $k \le r$.

If $\#F^{k-1}<k$, this particular debate satisfies the unanimity condition—i.e., all correct processors start $Debate(k,1)$, hence $E^k[\lambda$ is 1-common]. On the other hand, if $\#F^{k-1} \ge k$, then a k-favorable condition holds for $Debate_k$, with $Z=F^{k-1}$. Assume, by way of contradiction, that for correct processor i node σj belongs to U (the set of undecided nodes—see Fig. 3) in round r. Then σ is undecided, and by Lemma 2.8, all the processors in the sequence σ belong to F_i^r-Z, hence $\#(F_i^r-F^{k-1}) \ge |\sigma| = r-k > \#F_i^r - \#F^{k-1}$, a contradiction. Thus, U becomes empty in i's debate after round r, all correct processors have empty U after round $r+1$, and all of them have empty M after round $r+2$. This means, in particular, that $K_i^r[\lambda$ is 1-

common] holds, for $k \le r$.

Because a 0 vote by a correct processor in $Debate_{r+1}$ and $Debate_{r+2}$ would imply $\#F^{r-1} > r$, these debates satisfy the unanimity condition. By the same reasoning as in Lemma 2.9, all the correct processors start $Debate(l,1)$, for $l > r$. □

Besides increasing the patience threshold S to $2n^3$, we now introduce the following modifications to *MAIN* of Fig. 4 (we assume each processor has a boolean variable *Done* initialized to **false**):

- At the beginning of round k:

  if *Done* **then Send**('I am done');

  (processors which send 'I am done' are assumed to start subsequent debates with 1).

- At the end of round k:

  if *Done* **then halt**;
  Done := $\#F_i^k \le k$;

The only remaining situation to be handled is the early stopping of $Debate_0$, in the case $f = 0$. For this case, we note that unanimous messages in round 1 allow processors not to send messages in round 2.

The above allows us to state the following

Theorem 3.1: The *Cloture Votes* protocol solves the Distributed Consensus problem in $\min\{t+1, f+2\}$ rounds, where f is the actual number of processors that fail in the run, using messages of size $O(n^5)$, provided $n > 4t$.

4. Concluding Remarks

In this paper we have presented the first protocol to achieve Distributed Consensus with total number of processors $n > 4t$, optimal number of rounds and polynomial message size. The protocol is based on the new *Cloture Votes* technique, used in conjunction with the existing early stopping and fault masking techniques. These techniques can be applied and extended in several directions. One can study a wide variety of protocols by changing the resolution scheme of protocol *EIG* [BDDS] and the fault masking method, and introducing new rules of inference. The resulting round-optimal protocols have messages of size $O(n^4)$ for $n > 4t$, $O(2^{4/\varepsilon} n^5)$ for $n > (3+\varepsilon)t$, and $O(2^{\sqrt{n}})$ for $n > 3t$, respectively (see [BG2]). (One should note that for larger values of n the message size can be reduced even further by using other techniques: Moses and Waarts achieve $m = O(n^2)$ for $n > 8t$ [MW], while Coan and Welch have $m = 1$ for $n = \Omega(t \log t)$ [CW2].)

The message size can also be drastically reduced when the number of rounds is moderately increased, using the message size/number of rounds trade-off technique described in [BG1]. Thus, Distributed Consensus is possible with either polynomial message size, near optimal number of processors ($3t + O(t/\log t)$) and optimal number of rounds, or polynomial message size, optimal number of processors and near optimal number of rounds ($t + O(t/\log^2 t)$). Narrow gaps like these suggest that a "triply-optimal" protocol should be possible.

Acknowledgements

The authors would like to thank the Computer Science Department of the University of Chicago, for its hospitality during critical stages of this work. We are also thankful to the anonymous referees and to the editor, Ray Strong, for making many suggestions which helped us improve the presentation. Special thanks to Ray Strong for his encouragement throughout. This work was partially supported by AFOSR contract 87-0400 and NSF grant CR 8805978. The work of Juan Garay was also partially supported by the Leo S. Rowe Pan American Fund of the Organization of American States.

References

[BD] A. Bar-Noy and D. Dolev, "Families of Consensus Algorithms," *Proc. 3rd Aegean Workshop on Computing*, June/July 1988, pp. 380-390.

[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 Annual ACM Symp. on Principles of Distributed Computing*, pp. 42-51, August 1987.

[BG1] P. Berman and J.A. Garay, "Asymptotically Optimal Distributed Consensus," *Proc. 16th International Colloquium on Automata, Languages and Programming*, LNCS Vol. 372, pp. 80-94, July 1989.

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

[BGP] P. Berman, J.A. Garay and K.J. Perry, "Towards Optimal Distributed Consensus," *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 00. 410-415, October/November 1989.

[BL] J. Burns and N. Lynch, "The Byzantine firing squad problem," *Advances in Computing Research*, Vol. 4, pp. 147-161, 1987.

[C1] B. Coan, "A communication-efficient canonical form for fault-tolerant distributed protocols," *Proc. 5th. Annual ACM Symp. on Principles of Distributed Computing*, pp. 63-72, August 1986.

[C2] B. Coan, "Efficient agreement using fault diagnosis," *Proc. 26th Allerton Conf. on Communication, Control and Computing*, pp. 663-672, 1988.

[CW1] B. Coan and J. Welch, "Modular Construction of Nearly Optimal Byzantine Agreement Protocols," *Proc. 9th. Annual ACM Symp. on Principles of Distributed Computing,* pp. 295-306, August 1989.

[CW2] B. Coan and J. Welch, "Modular Construction of an Efficient 1-Bit Byzantine Agreement Protocol," *Mathematical Systems Theory*, this special issue.

[DRS] D. Dolev, R. Reischuk and H.R. Strong, "Early Stopping in Byzantine Agreement," *Journal of the ACM* Vol. 37, No. 4 (1990), pp. 720-741.

[DS] D. Dolev and H.R.Strong, "Polynomial Algorithms for Multiple Processor Agreement," *Proc. 14th Annual ACM Symp. on Theory of Computing,* pp. 401-407, May 1982.

[FL] P. Feldman and S. Micali, "An Optimal Probabilistic Algorithm for Byzantine Agreement" (invited paper), *Proc. 16th International Colloquium on Automata, Languages and Programming,* LNCS Vol. 372, pp. 341-378, July 1989.

[FL] M.J. Fischer and N. Lynch, "A lower bound for the time to assure interactive consistency," *Information Processing Letters,* 14:4 (1982), pp. 183-186.

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

[LSP] L. Lamport, R.E. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems,* Vol. 4, No. 3 (1982), pp. 382-401.

[MW] Y. Moses and O. Waarts, "Coordinated Traversal: ($t+1$)-Round Byzantine Agreement in Polynomial Time," *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pp. 246-255, October 1988.

[PSL] M. Pease, R. Shostak and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM,* Vol. 27, No. 2 (1980), pp. 228-234.

[TPS] S. Toueg, K.J. Perry and T.K. Srikanth, "Fast Distributed Agreement," *SIAM Journal on Computing,* Vol. 16, No. 3 (1987), pp. 445-458.