Early Stopping in Byzantine Agreement

DANNY DOLEV

Hebrew University, Jerusalem, Israel, and IBM Research, Almaden Research Center, San Jose, California

RUEDIGER REISCHUK

Institut fuer Theoretische Informatik, Technische Hochschule Darmstadt, Darmstadt, West Germany

AND

H. RAYMOND STRONG

IBM Research, Almaden Research Center, San Jose, California

Abstract. Two different kinds of Byzantine Agreement for distributed systems with processor faults are defined and compared. The first is required when coordinated actions may be performed by each participant at different times. This kind of agreement is called Eventual Byzantine Agreement (EBA). The second is needed for coordinated actions that must be performed by all participants at the same time. This kind is called Simultaneous Byzantine Agreement (SBA).

This paper deals with the number of rounds of message exchange required to reach Byzantine Agreement of either kind (BA). If an algorithm allows its participants to reach Byzantine agreement in every execution in which at most t participants are faulty, then the algorithm is said to tolerate t faults. It is well known that any BA algorithm that tolerates t faults (with $t < n - 1$ where n denotes the total number of processors) must run at least $t + 1$ rounds in some execution. However, it might be supposed that in executions where the number f of actual faults is small compared to t, the number of rounds could be correspondingly small. A corollary of our first result states that (when $t < n - 1$) any algorithm for SBA must run $t + 1$ rounds in some execution where there are no faults. For EBA (with $t < n - 1$), a lower bound of $\min(t + 1, f + 2)$ rounds is proved. Finally, an algorithm for EBA is presented that achieves the lower bound, provided that t is on the order of the square root of the total number of processors.

Categories and Subject Descriptors: C.2.4 **[Computer-Communication Networks]**: Distributed Systems—*distributed applications*; *distributed databases*; *network operating systems*; C.4 **[Performance of Systems]**: *reliability, availability, and serviceability*; F.1.2 **[Computation by Abstract Devices]**: Modes of Computation—*parallelism*; H.2.4 **[Database Management]**: Systems—*distributed systems*

General Terms: Algorithms, Reliability, Theory, Verification

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, early stopping, fault tolerance, reliability

---

Some parts of the paper are extended and improved versions of parts of papers that appeared in the *Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science* [8] and in the *Proceedings of the 2nd International Symposium on Distributed Data Bases* [11].

Authors' addresses: D. Dolev, Hebrew University, 91904 Jerusalem, Israel; R. Reischuk, Institut fuer Theoretische Informatik Technische Hochschule Darmstadt, 6100 Darmstadt, West Germany; H. R. Strong, IBM Research, Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

1. *Introduction*

In this paper, we discuss two closely-related types of agreement that can be reached in a distributed system in the presence of undetected processor faults. One type is called *Simultaneous Byzantine Agreement* (*SBA*) and the other *Eventual Byzantine Agreement* (*EBA*). Corresponding to these two types of agreement are two distinct problems in coordination among multiple processors in a distributed system. One problem is synchronization: Processors may be required to perform some action at the same time, immediately after reaching agreement on that action [18]. The other is consistency as required, for example, in the atomic commitment of a distributed database transaction. The participants in the transaction commit protocol must agree on whether or not the transaction is to be committed. In this case, it is enough to know that the choice will eventually be the choice of all other parties to the agreement [10]. (Note that the atomic commit problem that implies BA is the problem of nonblocking atomic commit with guaranteed communication. In this problem, correct participants must reach a decision in spite of the failure of any other participants or coordinators and without waiting for the recovery of others.)

It is the purpose of this paper to explore the difference between these two problems and the consequent differences in requirements for their solution. Because SBA implies EBA within our model, EBA can always be reached as early as SBA. We show that EBA can often be reached earlier than SBA.

The context for our study is a network of n processors that are able to conduct synchronized rounds of information exchange, each round consisting of message transmission, message receipt and processing. In the following, n will always denote the number of processors. We assume that the network is completely connected and that only processors can fail. The reader may be interested in exploring models with weaker assumptions in the following related references: [2], [4], [10], [17], and [23]. The reader might also like to explore earlier related work in [5], [6], [7], [14], [16], and [19]. In the Byzantine fault case, no assumption is made about the behavior of faulty processors. During an execution of an algorithm, a processor is said to be *correct* if it follows the specifications of the algorithm; otherwise, it is said to be *faulty*.

We assume that the agreement to be reached concerns a single value that is initially given as input to one processor, called the *origin*. This value is taken from a known set of values. All processors are assumed to know when the input is given to the origin. Each processor is to give exactly one *output* value after some number of rounds of information exchange with the other participating processors. The processors are said to have reached *agreement* when the following two conditions hold:

(i) all correct processors have given the same value as output, and
(ii) if the origin is correct, then all correct processors have given the input value as output.

Byzantine agreement was originally defined in [21] using these two conditions. We call such a state *eventual* agreement, emphasizing the fact that nothing is assumed about the relative times at which the correct processors give their output values. We say that the agreement is *simultaneous* if

(iii) all correct processors give their outputs at the same round.

When no assumption is made about the behavior of the faulty processors, we modify the term agreement with the adjective *Byzantine*. Thus, we have the terms

eventual Byzantine agreement (EBA) and simultaneous Byzantine agreement (SBA). A protocol or algorithm guarantees (Byzantine) agreement in some set of executions if, in each execution of the set, all correct processors reach a (Byzantine) agreement.

Note that a processor may give its output in one round and also continue to send messages to other processors in that and subsequent rounds. In this case, the processor has not finished all rounds of message exchange required by its algorithm when it gives its output. A processor is said to have *stopped* in round r, if it has given its output by round $r + 1$, and otherwise sends no messages in any round after r. In an execution of an algorithm for reaching agreement, we count the number of rounds between initial input and final stopping of all correct processors as the number of rounds *required* by the algorithm.

If an algorithm allows its participants to reach Byzantine agreement in every execution in which at most t participants are faulty, then the algorithm is said to *tolerate* t faults. In this paper, we investigate the number of rounds required to reach agreement as a function of the number of actual faults and the number of faults to be tolerated. Suppose A is an algorithm that tolerates $n - 2$ faults, requiring a maximum of k rounds. Let A' be the algorithm obtained by modifying A so that, no matter what happens, each processor stops after k rounds, the origin always gives as output its input value, and each other processor gives as output the value A would give, if any, or a default value, otherwise. Inspection of the definition of agreement shows that A' tolerates any number of faults. Hence, we assume $t < n - 1$, unless otherwise indicated. The initial work of Pease et al. [21] showed that agreement in the presence of up to t faults could be reached by round $t + 1$, provided the number of processors was sufficiently large. Later, $t + 1$ was shown to be a lower bound on the number of rounds required in the worst case [3, 9, 15]. A natural question arises from this worst case bound: Can an algorithm for agreement be constructed to handle up to t faults so that whenever the number f of actual faults is smaller than t, the number of rounds required to reach agreement is smaller than $t + 1$? Sections 2 and 3 present lower bounds for this problem.

First, in Section 2, we generalize a previous result [9] to show that $t + 1$ is a general lower bound for SBA for any $f \leq t$. More formally, we show that for any protocol that guarantees SBA in the presence of up to t faults, and for any number $f \leq t$, there exist executions of that protocol with only f actual faults in which correct processors send messages to other processors in round $t + 1$. In particular, whenever there are no actual faults, an SBA algorithm must run at least $t + 1$ rounds.

Later, Dwork and Moses extended this bound (first published in [12]) by studying a closely related problem in which each processor has an input [13]. A function from the set of faulty processors to integers that gives the round number at which each processor failed is called a *pattern*. Dwork and Moses give a lower bound on the number of rounds required for their problem as a function of the pattern. Their bound is easily shown to be a bound for our problem as well by choosing the worst pattern.

The $(t + 1)$-lower bound—and also that of Dwork and Moses—hold when the set of faults to be tolerated is restricted to a very simple type of fault called a *crash* fault. When a processor suffers a crash fault, it sends a subset of messages it is specified to send in one round and simply ceases to operate from then on. However, Theorem 2.1 even holds if the faulty behavior is further restricted to a class of faults called orderly crash faults (defined below).

Extending the proof method of Section 2 we show in Section 3 that EBA requires at least $\min(f+2, t+1)$ rounds. Our proof works only for crash faults and we do not know how to prove this result for orderly crash faults. Note that we are talking here about the worst case number of rounds required for any execution with only f faulty processors of an EBA protocol that must tolerate t faults. It is important to notice the difference between the time at which a processor chooses an output value and the time at which it can cease executing the protocol. For many restricted fault classes including crash faults, the output value can be obtained by round $f+1$ using a simple diffusion algorithm, but some processors must continue to send messages through round $f+2$.

In this paper, we count the number of rounds of information exchange required to complete the actions specified by the protocol, not the number of rounds required for all correct processors to have produced an output value. Since giving its output early cannot help a processor to stop earlier, we assume (for the rest of this paper) that a processor saves its output until the round after it last sends a message to another processor. This assumption is a notational convenience and is made without loss of generality. It is easy to convert any simultaneous agreement algorithm to one in which all correct processors stop before they give their outputs and outputs are given no later than in the unconverted algorithm. It is easy to convert an eventual agreement algorithm so that one round after every correct processor knows its output value, every correct processor has stopped.

In Section 4, we present an algorithm for EBA that achieves our lower bound, provided $n > \max(4t, 2t^2 - 2t + 2)$. This algorithm does not depend on any authentication protocol. It requires $\min(f+2, t+1)$ rounds to reach EBA using a polynomial (in both n and t) number of bits of information exchange. Previous early stopping EBA algorithms did not achieve the lower bound but did work for $n > 3t$. We refer the reader to our previous work [8], that of Toueg et al. [24], and that of Coan [1].

In the remainder of this section, we present the model for execution of an agreement algorithm that is used in both our lower-bound proofs and in the presentation of our algorithm. The model we present here is similar to the one previously given by Dolev and Strong [11]. The formal framework represents a round of an execution as a directed graph with labeled edges and nodes as follows.

Let V denote a set of possible values (including the values 0 and 1) and let MSG denote a set of possible messages. A *history* is an infinite sequence of rounds. Each *round* consists of a directed labeled graph with nodes corresponding to a set P of n participating processors, together with special source and sink nodes (that are not in P). There is an edge corresponding to every ordered pair of nodes. Each edge is labeled by an element of MSG (the message sent), an element of V (a value), or an empty label (indicating no message). For notational convenience, each history begins with round 0, in which the edge coming from the source outside P to the origin is labeled with the input value from V. All other edges have the empty label at round 0. At any subsequent round, any node may have the edge from it to the sink node outside P labeled with its output value. During this round and subsequently all other edges from this node, carry the empty label. If node p has such an edge to the sink at round k, then p has stopped (information exchange) at round $k-1$, and its output value is the value on the edge to the sink.

Messages (labels) on edges directed toward p in round k are said to be *received* by p at round k. Likewise, messages on edges directed from p in round k are said to be *sent* by p at round k. If H is a history we write pH for the *view of H according*

to p, which consists of the sequence of subgraphs of the rounds of H that have all the labeled nodes but only the edges that are adjacent to p. We also write H_k and pH_k for the *initial sequence* of H from its beginning through round k and its view according to p, respectively.

A *protocol* (or algorithm) A takes as input an initial subsequence of a view of a history according to a processor and produces an ordered set of labeled edges directed from that processor for the next round. Let $U(A, t)$ be the set of all histories on a fixed set of processors in which all correct processors follow A and in which at most t processors fail to follow A. (In each section, we restrict $U(A, t)$ to histories that have only failures of certain types. Also, we write U for $U(A, t)$ when the arguments A and t are clear from the context.)

A *t resilient agreement algorithm* is an algorithm A such that in each history of $U(A, t)$, each correct processor stops in some round and the processors reach agreement ((i) and (ii) above).

Note that a history includes the names of the processors, and the view of a history according to one processor is assumed to include the names of all its neighbors (in the completely connected network), whether they have sent it messages or not. Thus, an agreement algorithm need not be uniform: the actions it prescribes can depend on the name of the processor acting and on the names of its targets.

2. *The Lower Bound for SBA*

In Sections 2 and 3, we restrict attention to histories in which the only way a processor can fail to follow its algorithm is to fail to send some or all of its prescribed messages in one round and remain silent thereafter. This is the notion of a crash failure defined above. This notion is a close relative of the notion of a "fail-stop processor" [22]. Note that in the round in which a processor has a crash failure, it may send any subset of the messages specified to be sent at that round. It does not send any message at any subsequent round.

In proving the lower bound in this section, we further restrict the failure mode to *orderly crash* failures in which failing processors must respect the order specified by the protocol in sending messages to neighbors. (Recall that for each round a protocol produces an ordered set of labeled outedges that we identify with messages to be sent.) If a processor fails to send a specified message, it must also fail to send any message specified to be sent after that message in the protocol ordering.

A processor *fails* during the first round in which it does not send all messages required by algorithm A. A processor that fails in round r, sends no messages in each succeeding round.

Our lower-bound proofs are based on establishing certain equivalences among histories. The following definitions are crucial in following the proofs. Let A be an agreement algorithm that guarantees SBA in the presence of at most t orderly crash faults. Let P be a fixed set of n processors. Recall that $U(A, t)$ is the set of histories with only orderly crash faults in which algorithm A is employed by all correct processors, and the number of faulty processors does not exceed t. We introduce two equivalence relations on the set $U(A, t)$. These equivalences are also defined for the set of k round initial sequences of such histories, for any k.

The first is *witness equivalence*. For k round initial sequences, this is the transitive closure of the relation that holds between H_k and J_k when for some processor p correct in both, $pH_k = pJ_k$. Histories H and J are witness equivalent if their k round initial sequences are witness equivalent for every k. In other words, witness

equivalence (through round k) is the transitive closure of the relation that holds between two histories if there is a processor correct in both that cannot distinguish between the two (through round k).

The second is *output equivalence.* Here, we take the transitive closure of the relation that holds between H and J (or H_k and J_k) when some processor correct in both gives the same output value in both.

A history H is said to be *serial* if

(1) H is in U,
(2) for each positive integer k, $k \leq t$, the number of processors exhibiting faulty behavior in H_k does not exceed k, and
(3) no processor fails after round t.

Note that each history of U in which there are no faults is a serial history. Recall that a pattern (for a history) is a function from the set of faulty processors to integers that gives the round number at which each processor failed. We call one pattern a *subpattern* of another if the corresponding history for the first pattern has as faulty processors only a subset of that of the second and the first pattern is the corresponding restriction of the second. If a history H in U has a pattern of failures that is a subpattern of that of a serial history, then H is also a serial history.

If H_k is an initial sequence of a history in U, then the *conservative extension* of H_k is the unique history H' in U such that

(1) $H'_k = H_k$, and
(2) no processor fails after round k.

Given a processor p in a history H in U, the *silencing* of p at round k of H is the unique history H' (not necessarily in U) such that

(1) $H'_k = H_k$ except that p sends no messages in round k of H',
(2) no processor (except possibly p) fails after round k,
(3) p sends no messages after round k.

Its uniqueness is guaranteed because conditions (1) and (3) completely determine the behavior of p. For the remaining processors, observe that (2) forces all processors that are correct in H_k to follow A in all subsequent rounds and processors faulty by round k cannot send any messages after round k (we have restricted to crash failures). If history H has processors other than p that fail after round k, then H' resembles the conservative extension of H_k on those processors because they do not fail in H'. However, the silencing of p at k is not necessarily the conservative extension of its k round initial sequence because A may not call for p to send any messages in round k, but it might call for p to send messages later. Since we want p to remain silent from round k on, we must allow for the possibility that p fails in some round after round k. Note that if adding p to the set of faulty processors of H does not raise its cardinality above t, then the silencing of p at round k of H is in U.

A processor p is said to be a *candidate* in round i of a history H if p does not fail before round i and if both H and the silencing of p at round i of H are serial. Note that if p fails in round i of serial history H, then p is a candidate in round i of H. However, p can be both correct in H and a candidate in round i of H.

THEOREM 2.1. *If agreement algorithm A guarantees SBA for each history with at most t orderly crash faults, then A requires at least $\min(n - 1, t + 1)$ rounds to reach SBA in any serial history.*

PROOF. We base the proof on a sequence of lemmas that contains ideas from several previous related proofs [8, 11, 12]. Suppose for the rest of this section that algorithm A guarantees SBA for each history with at most t orderly crash faults. Assume that there is a serial history H in which A reaches SBA in fewer than $\min(n-1, t+1)$ rounds. If $t > n-2$, then A guarantees SBA for each history with at most $t' = n-2$ orderly crash faults and A reaches SBA in H in fewer than $n-1 = t'+1$ rounds. Thus, a counterexample with $t > n-2$ would provide a counterexample with $t' = n-2$. Hence, we assume (without loss of generality) that n is at least $t+2$.

LEMMA 2.2. *Let H and J be histories in U. If A uses k rounds to reach SBA in J and H_k is witness equivalent to J_k, then A uses k rounds to reach SBA in H, and H and J are output equivalent.*

The following straightforward, but long proof is supplied for completeness. The technical details would better be left as an exercise to the reader.

PROOF. Recall that witness equivalence is the transitive closure of the relation that holds between two histories when there is a processor correct in both and with the same view in both. If $pH = pJ$ and p is correct in both histories, then we call p a *witness* to the equivalence of H and J. Since witness equivalence is the transitive closure of the relation that holds where there is a witness, any two witness-equivalent histories are related as follows: If H and J are witness equivalent, then there exist finite sequences $\{H(i) \mid 1 \le i \le m\}$ and $\{p(i) \mid 1 \le i \le m-1\}$ such that $H(1) = H$, $H(m) = J$, and $p(i)$ is a witness to the equivalence of $H(i)$ and $H(i+1)$, for each i with $1 \le i \le m-1$. We say that H is witness equivalent to J *via* the sequence $\{p(i) \mid 1 \le i \le m-1\}$ of witnesses. This discussion applies equally to witness equivalence as a relation on initial sequences of histories.

The proof is a straightforward induction on w, the length of a sequence W of witnesses such that H_k is witness equivalent to J_k via W.

First, let $w = 1$, and let p be the witness. Let v be the output value that algorithm A specifies for p in J. Since A uses k rounds to reach SBA in J, v is determined by pJ_k. Since the agreement in J is simultaneous, all correct processors stop at round k in J. Thus, no correct processor sends any message to other processors at round $k+1$ of J. Processors correct through $k+1$, just give their output at that round.

Since witness equivalence of J_k and H_k does not give any condition on the correctness of the witness p in later rounds, one cannot simply compare J and H (p may become faulty in round $k+1$ when it is supposed to give its output). Instead, let J' be the history that is identical to J except that, if p does not remain correct in J at round $k+1$, then in J' p correctly gives its output v in round $k+1$. Obviously, J_k and J'_k are witness equivalent, since they are identical. The assumption $n > t+1$, implies that there is some $q \ne p$ that remains correct in J. History J' is in U and $qJ' = qJ$ for any correct q in J with $q \ne p$. Since A uses k rounds to reach SBA in J, q gives its output in round $k+1$ of J. Since $qJ = qJ'$, q gives the same output in round $k+1$ of J'. Thus, J and J' are output equivalent. Because J' is in U, A is guaranteed to reach SBA in J'. Thus, all correct processors give the same output v in round $k+1$ of J'.

Similarly, we define H' identical to H, except that p remains correct in H' through round $k+1$. Since by assumption J_k and H_k are witness equivalent, so are J'_k and H'_k. Therefore, as in J'_k, p in H'_k gives output v in round $k+1$. Since H' is in U, the argument above shows that all correct processors in H', give output v in round $k+1$, and that H' is output equivalent to H. Since p is correct in both

H' and J', we conclude that H' and J' are output equivalent. Since $n > t + 1$, there is some processor $q' \neq p$ that remains correct in H. Since $q'H' = q'H$, q' gives output v in round $k + 1$ of H. Since H is in U, A reaches SBA in H. Thus, all correct processors give output v at round $k + 1$ of H. This completes the proof of Lemma 2.2 for $w = 1$.

Suppose we have proved the lemma for witness equivalence via sequences with lengths at most w; and suppose that H_k and J_k are witness equivalent via a sequence of $w + 1$ witnesses. Then there is a history K in U such that K_k is witness equivalent to J_k via a sequence of w witnesses and H_k is witness equivalent to K_k via 1 witness. By the induction hypothesis, J and K are output equivalent and A uses k rounds to reach SBA in K. Also by the induction hypothesis, H and K are output equivalent and A uses k rounds to reach SBA in H. This completes the proof of Lemma 2.2 □

In the rest of the proof, we show how to alter serial histories in a way that preserves witness equivalence, but changes the number of faults and the place of their occurrence. In any history H of U in which p fails to follow algorithm A, there is a first message specified by A that p fails to send. Also in any round of H in U in which p sends any messages, there is a last message sent by p (in the order specified by A). We call an outedge e of p in a round of a history H *significant* if algorithm A specifies a message to travel over that edge and this message is either the last message sent by p in this round of the history or the first message specified by A in the entire history that p fails to send. Since we consider only orderly crash faults, the message on any significant edge is either correct or absent but not both. We show how to alter the states of messages on selected edges from absent to correct, or vice versa, producing witness equivalent initial sequences of histories and eventually producing a desired result. In particular, we are able to correct any faulty processor or cause any processor to fail in any round that does not violate the requirement that the resulting history be serial.

LEMMA 2.3. *If e is a significant outedge of a candidate p in round $k \leq t$ of a serial history H, then there is a serial history J such that J_t is witness equivalent to H_t and J_k is identical to H_k except that the state of the message at e is altered (from correct to absent or vice versa).*

PROOF. First note that if e is a significant outedge of a candidate p in round k of history H in U, then the operation of altering the state of the message on e in H_k and then taking the conservative extension of the altered initial sequence produces a serial history J. This is true because, by definition S, the silencing of p at round k of H, is serial and the pattern of J is a subpattern of the pattern of S. All processors other than p are either correct in both histories or faulty in both. If p is correct in H until round k, then k becomes faulty both in J and S at that round; otherwise, p has to become faulty in H in round k by not sending the appropriate message on e. In that case, p in S becomes faulty in S at round k, too. Either the same is also true for J or by adding a message to e, p now behaves correct in round k. Since J gets extended conservatively after that round in this case, p never gets faulty.

Let e be a significant outedge of candidate p in round k in history H in U. We prove the lemma by induction on $t - k$.

In case $t - k = 0$, in the initial subsequence H_t we simply alter e and take the conservative extension, producing a history J in U identical to H through round $k = t$, except for e. Since p is a candidate, J is serial. Note that, except possibly in

p, the correct processors in H and J are the same. Since $n > t + 1$ and there are at most t faulty processors, there is at least one processor q such that q is not the target of e and q is correct in both histories. Now $qH_t = qJ_t$ so H_t and J_t are witness equivalent. This completes the proof for $t - k = 0$.

Now assume the lemma holds for all rounds r with $(t - r) < (t - k)$. Let e be a significant outedge of candidate p in round k of history H in U and assume $k < r \leq t$. We next show how to correct all the processors that fail in rounds $k + 1$ to t of H.

Let q be any processor that fails in round r of serial history K in U. Then q is a candidate in round r of K. By successive application of the induction hypothesis to significant outedges of q with missing messages, there is a serial history L such that L_r is identical to K_r except for outedges of q in round r, L_t is a witness equivalent to K_t, and q is correct in L_t. Let L' be the conservative extension of L_t. Then L' is a serial history with L'_t witness equivalent to K_t, L'_r identical to K_r except for outedges of q in round r, and q correct throughout L'. Thus, by successive application of this principle to processors that fail after round k in H, there is a serial history M in U such that M_t is witness equivalent to H_t and M is the conservative extension of H_k.

Since $k < t$ and M has no failures after round k, any processor correct in M is a candidate in round $k + 1$ of M. Let q be the target of e. If q fails in some round of M and N is the conservative extension of H_k with e altered, then for any correct $q' \neq p$ in M, $q'M = q'N$, so M and N are witness equivalent and N is the desired history. Otherwise, assume q is correct in M. By successive application of the induction hypothesis to any significant outedges from q in rounds from $k + 1$ to t of M, there is a serial history G such that G_t is witness equivalent to M_t, G_{k+1} is identical to M_{k+1} except for outedges of q in round $k + 1$, and q sends no messages in rounds from $k + 1$ to t of G. If p fails in round k of H, then p sends no messages after round k of G, so altering the state of the message on the edge from p to k in round k of G produces the desired history.

The case left to check is the one in which p does not fail in round k of H. In this case, p does not fail in round k of G and G_{k+1} has at most k faulty processors including q. Thus, p is a candidate in round $k + 1$ of G.

By successive application of the induction hypothesis to any significant outedges of p in rounds from $k + 1$ to t of G, there is a serial history F in U that is witness equivalent to G through round t, identical to G through round k, and has no messages from p or q in rounds from $k + 1$ to t. Let E be the conservative extension of the result of altering the state of the message on edge e (the edge from p to q) in round k of F_t. The only possible changes to the pattern of F are moving the failure of p as early as round k and the failure of q as early as round $k + 1$. We have already established that this pattern is that of serial history so E is serial. Since $n > t$, there is a processor q' other than p or q correct in both E and F. For such a processor, $q'E_t = q'F_t$, so E_t is witness equivalent to F_t. Finally, E_k is identical to H_k except for the state of the message on e. Thus, E is the desired history. This completes the proof of the lemma. □

LEMMA 2.4. *If H and J are serial histories, then H_t is witness equivalent to J_t.*

PROOF. Let H be a serial history. By successive applications of Lemma 2.3 to significant outedges with missing messages from faulty processors of H, there is a serial history H' such that H'_t is witness equivalent to H_t and H' has no faulty processors. By successive applications of Lemma 2.3 to significant outedges of the origin in round 1 of H', H'_t is witness equivalent to N_t where N is the silencing of

the origin in round 1. Likewise, any other serial history J is witness equivalent through round t to N_t. □

To finish the proof of Theorem 2.1, consider the assumed history H, in which A reaches SBA in t or fewer rounds. Let v be the output value of the correct processors in H, let v' be a value different from v, and let J be the fault free (serial) history with input v'. By the agreement condition, all processors have to output v' in J. On the other hand, by Lemma 2.4, H_t and J_t are witness equivalent. By Lemma 2.2, H and J are output equivalent. This means that in J all processors had to output v, a contradiction. □

COROLLARY 2.5. *Algorithm A requires at least* $\min(n-1, t+1)$ *rounds to reach SBA when there are actually no faults.*

3. *A Lower Bound for EBA*

Next, we consider the question of early stopping for EBA and prove a lower bound similar to, though stronger than, the one in [11]. In this section, we restrict attention to histories in which all failures are crash failures. Let A be a t-resilient agreement algorithm that is supposed to guarantee EBA in $U(A, t)$. Note that $U(A, t)$ has a different definition in this section: faults in histories of $U(A, t)$ may be crash faults rather than the orderly crash faults of Section 2. When we refer to a conservative extension in this section, we mean a history defined as in the previous section but with respect to the current U.

An edge e in round k of history H is *critical* if there is a history J in U such that

(1) J is not output equivalent to H,
(2) J is identical to H through round k except for edge e, and
(3) J is the conservative extension of J_k.

In other words, an edge is critical if altering the state of its message and taking the conservative extension alters the output value of correct processors. Note that A must specify a message for any critical edge.

For this section, we require versions of the notions of serial and candidate that are parameterized by f. A history H is said to be *f-serial* if H is in U, H has no more than f faults, for each positive integer $k \leq f+1$, the number of processors exhibiting faulty behavior in H_k does not exceed k, and no processor fails in H after round $f+1$. A processor p is said to be an *f-candidate* in round i of history H if p does not fail before round i, and if both H and the silencing of p in round i of H are f-serial.

THEOREM 3.1. *Let A be an agreement algorithm that reaches EBA in histories of* $U(A, t)$. *Then there is a history in* $U(A, t)$ *with only f faults in which A requires at least* $\min(n-1, t+1, f+2)$ *rounds to reach EBA.*

PROOF. As we argued in the proof of Theorem 2.1, a counterexample with $t > n-2$ would provide a counterexample with $t = n-2$. Thus, we assume (without loss of generality) that $t < n-1$. Suppose that algorithm A reaches EBA within $\min(t, f+1)$ rounds in every history of U with at most f faults.

First, we give a straightforward derivation of a contradiction in the case $f = 0$. Assume A is a t-resilient agreement algorithm that uses only $\min(t, 1)$ rounds to reach EBA in any history of U with no faults. If $t = 0$, then processors send no messages to other processors; otherwise, when there are no faults, processors send messages to other processors only in round 1 and all processors give the input value as output in round 2. Let H_0 be the preliminary round that gives input 0 to

the origin and let H be its conservative extension. Each correct processor of H must give output 0 in round $\min(t + 1, 2)$. Let K_0 be the preliminary round that gives input 1 to the origin and let K be its conservative extension. Each correct processor of K must give output 1 in round $\min(t + 1, 2)$. In at least one of H and K the origin must send at least one message in round 1, for otherwise any processor except the origin would have identical views in the two histories. Thus, t must be greater than 0. Without loss of generality, assume that the origin sends a message to processor p in round 1 of H.

Let J_1 be identical to H_1 except that the origin fails in J_1 after sending only its message to p and let J be the conservative extension of J_1. (If the origin sends only one message in round 1 of H, then let $J = H$.) Then, J has at most one crash fault and is a history in U. Now $pH_1 = pJ_1$ so p gives output 0 in round 2 of both H and J. Thus, any correct processor in J must eventually give output 0. Since $t > 0$ and $n - 1 > t$, we have $n > 2$. Hence, there is a processor q that is neither the origin nor p. If the origin sent no message to q in round 1 of K, then we would have $qK_1 = qJ_1$. But q gives output 1 in round 2 of K and q gives output 0 in some round of J. Therefore, the origin must send a message to q in round 1 of K. Let L_1 be identical to K_1 except that the origin fails in round 1 by sending only its message to p (if any), and let L be the conservative extension of L_1. Then L has one crash fault and is a history in U. Since $pK_1 = pL_1$, p gives output 1 in round 2 of K and L, so any correct processor of L must eventually give output 1. Since p sends no messages to other processors after round 1 in any of the histories H, J, K, and L, we have $qJ = qL$. But this contradicts the fact that q must output 0 in J and 1 in L.

Now we assume $f \geq 1$. Since we assume $n - 1 > t$, there are at least two correct processors in any history of U. In any history of U with at most f faults, there can be no critical edge in round $\min(t, f + 1)$, because all correct processors have stopped by round $\min(t, f + 1)$ (giving their outputs by $\min(t + 1, f + 2)$) and changing a value over any single edge cannot affect the output of more than a single correct processor. We first show that in any f-serial history, there is no critical edge in round f from a processor that is an f-candidate in round f. Then, we show that all f-serial histories, including all histories with no faults, are output equivalent. As in the proof of Theorem 2.1, we then argue that histories with distinct inputs and no faults must have the same outputs, contradicting part (ii) of the definition of agreement. This contradiction will complete the proof. □

LEMMA 3.2. *Let H be an f-serial history. Then there is no critical edge in round f from a processor that is an f-candidate in round f of H.*

PROOF. We argued above that there could be no critical edge in round $\min(t, f + 1)$; so we can assume, without loss of generality, that $f < t$. Suppose there were a critical edge e in round f from a processor p that is an f-candidate in round f. Let q be the target of e. Let J be the conservative extension of the result of altering e in H_f. By the definition of critical edge, J is not output equivalent to H. Since p is an f-candidate in round f of H, and since H is f-serial, J is f-serial. Thus, J has no more than f faults and J is in U. Since we have assumed A reaches EBA within $\min(t, f + 1)$ rounds in every history of U with at most f faults, J reaches EBA by round $f + 1$.

Since $f + 3 \leq t + 2 \leq n$, we can find processors r and s correct in both H and J and not equal to q. Let H' be the conservative extension of the result of removing any message from q to r in round $f + 1$ of H_{f+1}. Let J' be the conservative extension of the result of removing any message from q to r in round $f + 1$ of J_{f+1}. Now H'

and J' each have at most t faults so that Algorithm A eventually reaches EBA in both. But each correct processor except r has the same view in H' as in H through $f+1$ and is stopped at $f+1$. By the view of s, $sH_{f+1} = sH'_{f+1}$ and therefore H and H' are output equivalent. Since H' is a conservative extension of an $f+1$ round initial sequence, no processor fails after round $f+1$ and no processor except r sends a message to another processor after round $f+1$. Likewise, each correct processor except r has the same view in J' as in J through $f+1$ and is stopped by $f+1$. Since $sJ_{f+1} = sJ'_{f+1}$, J and J' are output equivalent. Since J' is a conservative extension of an $f+1$ round initial sequence, no processor fails after round $f+1$ and no processor except r sends a message to another processor after round $f+1$. Thus, $rH' = rJ'$ and r must have the same output value in both, contradicting the assumption that H is not output equivalent to J. □

LEMMA 3.3. *If A reaches EBA for all histories with at most t faults and if A reaches EBA within $\min(t, f+1)$ rounds for all histories with at most t faults, then all f-serial histories are output equivalent.*

PROOF. Here we use the proof technique developed for Theorem 2.1. First, we show that if e is an outedge of an f-candidate p in round $k \leq \min(t, f+1)$ of f-serial history H, and if A specifies a message for e, then there is an f-serial history J output equivalent to H and identical to H through round k, except that the state of the message at e is altered (from correct to absent or vice versa). As in the proof of Lemma 2.3, this is proved by induction on $f+1-k$. Note that if e, H, and k are as above, then the conservative extension of the result of altering the state of the message on e in round k of H_k is an f-serial history and satisfies all requirements for J unless e is critical.

First, assume $f+1-k=0$, that is, $k=f+1$. Let e be as above in round $f+1$ of f-serial history H. Since there are no critical edges in round $f+1$, the appropriate conservative extension is the desired output equivalent history.

Next assume $f+1-k=1$, that is, $k=f$. Again the appropriate conservative extension is the desired output equivalent history by Lemma 3.2.

Finally, assume that we can obtain the desired output equivalent history for such edges in rounds r with $r > k$ and assume $k < f$. By successive application of the induction hypothesis to outedges from processors that fail in rounds after round k, we can obtain an f-serial history K that is output equivalent to H such that K is the conservative extension of H_k. Since $k < f$ and K is f-serial, any correct processor of K is an f-candidate in round $k+1$ of K. Now either the target q of e fails by round k or it is correct in K. If it is correct in K, then the silencing of q in round $k+1$ of K is also f-serial and can be shown to be output equivalent to K by successive applications of the induction hypothesis to the outedges of q. Thus, in any case there is an f-serial history K' that is identical to H through round k, is output equivalent to H, and has no messages from q after round k. Let J be the conservative extension of the result of altering the state of the message on e in K'_k. Then, J is output equivalent to K' because $rJ = rK'$ for any correct r in both J and K' and some such r exists because the source of e is the only possible additional fault in J over those in K'. Thus, J is the desired f-serial history output equivalent to H and identical to H through round k except for the state of the message on e.

The output equivalence of all f-serial histories follows easily by application of a proof analogous to that of Lemma 2.4. Let H be the f-serial history with input 0 and no faults. Let J be the f-serial history with input 1 and no faults. We have shown that H and J must be output equivalent, contradicting part (ii) of the

definition of agreement. Thus, A cannot reach EBA within $\min(t, f + 1)$ rounds in every history of U with at most f faults. This completes the proof of Theorem 3.1. □

4. *An EBA Algorithm*

In this section, we describe an algorithm for eventual Byzantine agreement that achieves the lower bounds of the previous sections, provided that n is sufficiently larger than t. The algorithm will tolerate up to t Byzantine faults. (We no longer restrict attention to crash faults.) The key to understanding this algorithm is the notion of separation, which will be described more formally below. Informally, when a faulty processor sends different information to two subsets of correct processors, it separates one set from another. The algorithm keeps track of two rounds of information exchange at a time, so a fault that separates large enough sets of correct processors from each other in one round will be discovered by all correct processors in the next round. In order to avoid discovery by all correct processors, a fault may only separate from others a set of the size of the number of unknown potential faults that must be tolerated. Thus, t faults cannot separate more than t^2 correct processors from other correct processors without at least one of them being discovered. The idea behind the algorithm is that when n is larger than $\max(4t, 2t^2 - 2t + 2)$, our algorithm will allow correct processors to obtain the agreement value at the end of any round in which no fault gives itself away and to stop within one additional round.

Recall that we count only the *rounds of information exchange* among the processors. The preliminary input and final output rounds are only used to simplify the description of the algorithm.

We use the following notation:

P denotes the set of names of participating processors,
s the name of the origin,
X a symbol not in P and
V the set of possible input values.
Let 0 be an element of V, let * a special value not in V (representing "undefined"), and let V' be the union of V and $\{*\}$.

To run the algorithm, each processor maintains a data structure consisting of two types of variables: variables containing values from the set V', and variables containing sets of processor names. For each of the strings s, ps, and pqs, where p and q run over all the elements of P, we associate a variable of the first type. The values stored in these variables will be interpreted as representing information received from the appropriate processors. Thus, for example, the value stored in s will be interpreted as the value sent by the origin of the agreement. The value stored in qs will be interpreted as the value q said that s sent to it. Finally, the value stored in pqs will be interpreted as the value p said that q said that s sent to it. Notice that s denotes both the origin and the variable associated with it. The pseudocode of the algorithm uses s only as a variable and not as a name for the origin.

With the string X and with strings pX, for every p in P, we associate a variable of the second type. Values stored in strings ending in X will be interpreted as representing information received from the processors about faults. Thus, the set

stored in X will be a set of processors known to be faulty. The set stored in qX will be the set of processors q claims to be faulty.

We refer to the variables as strings.

Strings ending in s will be initialized to value 0.
Strings ending in X will all be initialized to the empty set.

We use the following convention for naming sets of strings:

Let Q be any subset of P, let p be in P, and let R be any name for a set according to this convention. Then

$$Qs = \{qs \mid q \text{ is in } Q\},$$
$$QX = \{qX \mid q \text{ is in } Q\},$$
$$pR = \{pr \mid r \text{ is in } R\}, \text{ and}$$
$$QR = \{qr \mid q \text{ is in } Q \text{ and } r \text{ is in } R\}.$$

Thus, for example, Ps is the set of strings of length 2 that end with s, and pPs is the set of strings of length 3 that begin with p and end with s.

Here, we introduce a simple one round process that is the heart of many agreement algorithms. We give this process the name ROUND. Each processor executes ROUND during every round from round 3 until it stops. We also introduce a variant of ROUND called ROUND2 that is executed in round 2 and collects the original information in Ps. ROUND has two functions: (1) to exchange information on Ps with all other processors to produce values for PPs that are then reduced to values for Ps; and (2) to exchange information on X with all other processors to produce values for PX and to use PPs and PX to discover faults. It is expected to operate synchronously with all participating processors sending information to all and then receiving information from all. If two processors are correct, it is assumed that their information is correctly exchanged. It uses two auxiliary processors, DETECT and REDUCE, which are defined below. We assume that a processors sends messages to itself and processes them as part of all the messages it receives.

Note that in ROUND2 each processor sends the value it has stored in s and receives the corresponding values from all processors. It stores the value received from processor p in ps. Thus, ROUND2 has the instruction "RECEIVE ps from each p in P."

The action of each participating processor executing ROUND2 is as follows:

```
ROUND2:  /* for round 2 */
begin;
  SEND s to all processors;
  RECEIVE ps from each p in P;
    if ps is not received from p then set ps := s)
  if Ps does not contain at least n − t identical values then put the origin in X;
end ROUND2.
```

Note that in ROUND each processor sends the values it has stored in Ps and X to all processors and then receives corresponding values from every processor. The values received for Ps and X from processor p are stored in pPs and pX, respectively. Thus, ROUND has the instruction, "RECEIVE pPs, pX from each p in P."

The action of each participating processor executing ROUND is as follows:

```
ROUND:  /* for rounds after round 2 */
begin;
  SEND Ps, X to all processors;
  RECEIVE pPs, pX from each p in P;
    (if p is already in X
      then default its values for pPs to 0)
    (if p is not already in X but it does not send pPs and pX
      then for each q in P set pqs := * and leave pX unchanged)
  DETECT;
  for each p and q in P if pqs = * then set pqs := s;
  REDUCE;
end ROUND.
```

A correct processor may put the name of the origin in X during the execution of ROUND2, but only if Ps does not contain $n - t$ identical values so that the origin must be faulty. In later rounds, the process DETECT is the only way correct processors add names to the set of known faulty processors kept in X. DETECT is designed so that correct processors will never add names of correct processors to X, and therefore, at any time the largest possible number of faulty processors that a given correct processors has not discovered is $t - |X|$.

Since correct processors may stop at different times—the difference can be at most one round as will be seen later—one has to take care that a correct processor that has already stopped and therefore does not send messages anymore is not considered to be faulty. This is achieved by first setting variables pqs for which no value from p has been received to the undefined value "*". If p is not found faulty by DETECT, then pqs will later be set to the actual value of s.

If more than $t - |X|$ processors claim that they have put processor q in their set of known faulty processors, then any correct processor can safely put q in X (some other correct processor put q in its X first).

In our algorithm, correct processors send identical data to all participants. A property that will be preserved by REDUCE is that if p, q, and r are correct processors, then the values stored in pqs and rqs by any correct processor will be identical. Thus, if the multiset of values stored in $(P - X)qs$ does not have at least $n - t$ identical values, then q must be faulty.

The action of each participating processor executing DETECT is as follows:

```
DETECT:
begin;
  for each q in P − X;
    if |{p | p is in P − X and q is in pX}| > t − |X|
      or P − X contains two sets A and B each of cardinality ≥ t
        such that Aqs and Bqs both have only values in V,
        but no value occurs in both Aqs and Bqs
      then add q to X and default the values of qPs to 0;
    end for each q;
end DETECT.
```

The process REDUCE uses values of PPs to update the values of Ps using a majority vote. Let g be the smallest integer greater than $n/2$. In order to obtain the new value for string ps, a majority vote is taken over the values of the strings pPs. Note that all these values are obtained directly from p. There is no voting by others here on what p said as it is done by DETECT for q. If p is correct, then it sends the same data (Ps) to each participant; all correct participants will have the same value for ps after REDUCE. These values ps determine the further action to be

taken by each processor. If correct processors all have the same set Ps, then they behave identically and reach agreement very quickly.

The action of each participating processor executing REDUCE is as follows:

```
REDUCE:
begin;
  for each p in P;
    if pPs has at least g strings with value v
      then ps := v
      else ps := 0;
  end for each p;
end REDUCE.
```

For the remainder of this section, we assume that

$$n > \max(4t,\ 2(t + (t-1)^2)),$$

so that the following properties are true of the majority threshold g:

(1) $2g > n$;
(2) $n - 2t \geq g$;
(3) $n - t - (t-1)^2 \geq g$.

We use these properties of g to show that undetected faults cannot cause correct processors to reach different values for s.

The algorithm will be called EAGREE. It takes a value as input in round 0. Only the origin is given a value. If no value is received, the string s is left with its initial value 0. We use the existence of a value other than 0 stored in s in round 0 to indicate that the processor executing the code is the origin. All processors execute the same code. If a processor has a value other than 0 stored in s at the end of round 0, then it sends that value to all processors in round 1. We assume that no processor except the origin can have a value stored in s other than 0. If the input value is 0, the origin acts just like the other participants and sends nothing. Receiving nothing from the origin in the first round is interpreted as receiving 0 from the origin. This is just a convenience, all processors know the name (s) of the origin. This simply allows us to write EAGREE in a uniform way without mentioning explicitly the name of the processor executing the code. Correct processors ignore any values received from processors other than the origin in round 1. Correct processors using EAGREE reach EBA by round $\min(f+2,\ t+1)$. At the end of the algorithm, the variable s at each correct processor will hold the output value. Note that round 0 and the output round involve no information exchange among the processors and are not counted when we discuss the number of rounds required to reach agreement.

The action of each participating processor executing EAGREE is as follows:

```
EAGREE:
  begin;
  i := 0;  /* round 0—the input round */
    RECEIVE s AS INPUT;
    (if nothing is received, leave s unchanged)
  i := 1;  /* round 1 */
    if s ≠ 0 then SEND s to all processors;
    RECEIVE s;
      (if nothing is received from the origin, leave s unchanged)
  do i := 2 to t + 1;
    if i = 2 then ROUND2 else ROUND;
```

```
    if Ps has at least g identical values v
      then s := v;
      else s := 0;
    if Ps has at least n − t identical values
      then leave this do loop;
    end do;
  i := i + 1;  /* output round for this processor */
    OUTPUT s;
  end EAGREE.
```

Recall that $2g > n$ so that this algorithm is well defined.

THEOREM 4.1. *Execution of EAGREE by $n > max(4t, 2(t + (t-1)^2))$ processors results in EBA within $min(f + 2, t + 1)$ rounds, where f, the actual number of faults, does not exceed t.*

The proof of the theorem will be provided in the following series of lemmas.

LEMMA 4.2. *Suppose no correct processor is stopped at round $i - 1$ and let p, q, and r be correct processors. Then, at the end of round i, no correct processor has the name of a correct processor in X, every correct processor has $pqs = rqs$, and all correct processors share the same value for qs.*

PROOF. Suppose otherwise and let i be the first round in which the above conditions are violated in some execution of EAGREE. Since X and Ps are unchanged until round 2 and PPs is unchanged until round 3, it must be that $i \geq 2$. If $i = 2$ and some correct processor put the name of the origin in X, then this processor must not have received $n - t$ identical values in round 2. But if the origin had correctly either remained silent or sent identical values to all correct processors in round 1, then each correct processor would have received at least $n - t$ identical values in round 2. Thus, in this case, the origin must have been faulty. In round 2, only the origin can be put in X. Hence, no correct processor put the name of a correct processor in X in round 2. Each correct processor sends identical data to every processor, so if q is correct, each correct processor shares the value for qs in round 1. Moreover, in round 2, no correct processor will change the variables PPs from their initial value 0, hence $i \geq 3$.

Processors p and r receive the same set of values from a correct processor q in each round $j \leq i$, since by assumption no correct processor stops before round i. Further, we have supposed that no correct processor has the name of a correct processor in X at the end of round $i - 1$, thus neither p nor r will default the corresponding variables for q. Therefore, they share qPs at the beginning of the execution of DETECT in round i. Moreover, any correct processor not stopped at round 2 has the origin in X. Thus, for any processor not stopped at round $i - 1$, any subset of $P - X$ of cardinality $\geq t$ contains the name of at least one correct processor. There are two ways a correct processor adds a name to X in round $i > 2$. Both happen only during execution of DETECT. Since no correct processor had the name of a correct processor in X at the end of round $i - 1$, the first method (based on other processors claiming to have put the name in X) cannot introduce the name of a correct processor in round i. The second method for putting q in X requires correct processors p and r not stopped at round $i - 1$ with $pqs \neq rqs$. But this condition cannot hold for a correct q at the beginning of the execution of DETECT in round i. Thus, DETECT cannot have caused the failure of our conditions in round i. This means that no names of correct processors are included in X and that no correct processors q have their values for qPs defaulted to 0 at the

beginning of the execution of REDUCE in round i. Since REDUCE does not alter the values of *PPs*, and since all correct processors share their values for *qPs* for each correct q, the conditions cannot have been violated. □

We say that value v is *persistent* at round i, if at least g correct processors have v stored in s at the end of round i. Recall that a processor is said to stop in round i if the only action it takes in round $i + 1$ is to output its value. We say that a processor *is convinced* in round i if it has at least $n - t$ identical values stored in *Ps* at the end of round i. Note that if a correct processor is convinced in round i, then it stops in round i. Also, if a correct processor stops in round $i < t + 1$, then it is convinced in round i. However, a processor may stop in round $t + 1$ without being convinced. In this case, it gives its value for s as output without having $n - t$ identical values in *Ps*.

LEMMA 4.3. *If a correct processor is convinced at round $i \leq t + 1$, then the value it has for s must have become persistent by round $i - 1$.*

PROOF. Assume i is the first round at which some correct processor p is convinced, let us say of value v. Processor p must have at least $n - 2t$ strings *qs* with value v such that q is a correct processor. Let q be any such processor. By Lemma 4.2, q cannot be in X for p at round i. If $i = 2$, then q actually sent v to p in round i, so q had value v for s at the end of round $i - 1$. Since this holds for at least $n - 2t \geq g$ correct processors, v is persistent at round 1.

Assume now $i > 2$: Then before executing REDUCE in round i, p had values *qPs* such that either at least a majority (g) were v or there was no majority value implying $v = 0$. Processor q actually sent the values for *qPs* to p. These were the values q had for *Ps* at the end of round $i - 1$ and q sets s to v at this round. In any case, at least $n - 2t$ correct processors q had value v for s at the end of round $i - 1$; therefore, v became persistent. □

LEMMA 4.4. *If a value becomes persistent before round $t + 1$, then it remains persistent throughout the execution of the algorithm and is given as output by each correct processor. If a value becomes persistent before round t, then all correct processors are convinced at most two rounds later.*

PROOF. Consider the first round i at which any value v becomes persistent. Suppose $i < t + 1$. By Lemma 4.3, no correct processor stopped at round i. Let G be a set of at least g correct processors, each of which associates v with s after round i. By Lemma 4.2, no correct processor has any element of G in X at the end of round $i + 1$. Thus, at the end of round $i + 1$, each correct processor has each string of *Gs* valued v; so each correct processor has s valued v. If correct p and r are not stopped at round $i + 1$, then they send the same value for *qs*, for each correct q, in round $i + 2$. Thus, no correct processor adds the name of a correct processor q to X in round $i + 2$.

If any correct processor q' stops at round $i + 1$, then any correct processor not stopped at round $i + 1$, sets each string of q'*Ps* to v just before the REDUCE of round $i + 2$. Thus, each correct processor not stopped at round $i + 1$ has at least $n - t$ strings of *Ps* valued v and is convinced at the end of round $i + 2$. Each correct processor either stops at round $i + 1$ or round $i + 2$ and each gives output v.

For the second claim, suppose value v first becomes persistent at round $i < t$. By Lemma 4.3, no correct processor stops before round $i + 1$. As argued above, every correct processor has $s = v$ at round $i + 1$. Any correct processor not stopped at round $i + 1$ will see $n - t$ values v in *Ps* and will be convinced at round $i + 2$

because the strings of stopped processors default to $s = v$. We have already noted that if a correct processor stops at a round before round $t + 1$, then it is convinced at that round. Thus, each correct processor is either convinced at round $i + 1$ or at round $i + 2$. □

LEMMA 4.5. *If the origin is correct, then all correct processors will output its value.*

PROOF. If the origin is correct, then its value becomes persistent at round 1 and all correct processors output its value by Lemma 4.4. □

In order to keep any value from becoming persistent in a round, the faults must send distinct sets of values *Ps* to different sets of the correct processors. In fact, these sets *Ps* must reduce to distinct values. We say that a fault p *separates* sets A and B of correct processors if it sends them sets *Ps* so that after REDUCE, no member of A has a value stored in *ps* that is the same as that of a member of B. We call any set of correct processors a *witness set* if its cardinality is at least t and at most $n - 2t$.

LEMMA 4.6. *If for some i, $2 \le i \le t$, a fault p separates a witness set from all other correct processors at round i, and if some correct processor is not convinced by round $\min(i + 2, t + 1)$, then there are correct processors that do not have p in their set X by the end of round i, but by round $i + 1$ each correct processor will have p in X and set values pPs and ps to the default value* 0.

PROOF. Since some correct processor was not convinced by $\min(i + 2, t + 1)$, no value was persistent at round 1 by Lemma 4.4. Since $g + t \le n - t$, no correct processor could have $n - t$ identical values in *Ps* during the execution of ROUND2; so all correct processors put the origin in X at round 2. Thus for each correct processor, the origin is in X by the beginning of round $i + 1$. Note that, by Lemmas 4.3 and 4.4, no correct processor stopped at round i. Let A be the witness set separated from the rest of the correct processors by p in round i. Let B be the rest of the correct processors, which will be of size at least t. In round $i + 1$ before DETECT, each correct processor has values in *Aps* different from any values it has in *Bps*. Also, by Lemma 4.2, each correct processor has A and B as subsets of $P - X$. Thus, if p is not already in X, DETECT will add p to X and set *pPs* to 0, and REDUCE will then set *ps* to 0. But if p is in X for some member of A at the end of round i, then p is not in X for any member of B at the end of round i; so some correct processors do not have p in X at the end of round i. □

LEMMA 4.7. *If there is a correct processor that is not convinced by round $i + 2$ with $1 \le i \le t - 1$, then there is a set $\{p_j \mid 1 \le j \le i\}$ of i distinct faulty processors such that, for each j, each correct processor has p_j in X and value p_js defaulted to* 0 *by the end of round $j + 1$ (and in each succeeding round).*

PROOF. The proof is by induction on i. In case $i = 1$, let the origin be p_1. If some correct processor is not convinced by round 3, then, by Lemma 4.4, no value was persistent in round 1. As we argued in the proof of Lemma 4.6, each correct processor has p_1 in X and $p_1 s$ defaulted to 0 by the end of round 2. Once a processor p is in X, its values for *pPs* are defaulted to 0 in ROUND so REDUCE keeps the value 0 in *ps*. Thus, p_1 remains in X and $p_1 s$ remains defaulted to 0 after round 2.

Assume that the result has been shown for $i - 1$, with $2 \le i \le t - 1$ and assume that some correct processor is not convinced by round $i + 2$. By the induction

hypothesis, there are $i - 1$ distinct processors p_j for $1 \leq j \leq i - 1$ such that every correct processor has p_j in X and $p_j s$ defaulted to 0 by the end of round $j + 1$ and in every succeeding round. It suffices to show that there is a processor p_i such that p_i is distinct from p_j for $j < i$ and each correct processor has p_i in X and $p_i s$ defaulted to 0 by the end of round $i + 1$. (The argument above shows that these conditions continue to hold thereafter.) By Lemma 4.4, no value was persistent by round i.

If at round i, p separates a witness set from all other correct processors, then, by Lemma 4.6, at round $i + 1$ for each correct processor, p will be added to X and its values pPs and ps will be set to the default value 0. Note that no p_j with $j < i$ can separate correct processors at round i, since all of its values are defaulted to 0.

Suppose no fault separates a witness set from the other correct processors. Then, each of the at most $t - 1$ faults (not counting the origin) can separate at most $t - 1$ correct processors from the other correct processors. Hence, there would be a set of correct processors of size at least $n - t - (t - 1)^2 \geq g$ that agreed on s after round i and their value would be persistent. Thus, some fault separated a witness set from other correct processors at round i and this fault can be taken to be p_i. □

LEMMA 4.8. *If there are only $f < t$ faults, then all correct processors are convinced by round $f + 2$.*

PROOF. Assume to the contrary that there is a correct processor not stopped by round $f + 2$. By Lemma 4.7, there is a set $\{p_j \mid 1 \leq j \leq f\}$ of f distinct faulty processors such that, for each j, each correct processor has p_j in X and value $p_j s$ defaulted to 0 from round $j + 1$ on. Since there are exactly f faults, each fault is in X for each correct processor by the end of round $f + 1$. By Lemmas 4.3 and 4.4, no correct processor is stopped by round $f + 1$. But no fault is left to separate any correct processors, so each correct processor has identical values for PPs after round $f + 1$. Thus, in round $f + 2$ all correct processors send identical values for Ps and, by the end of round $f + 2$, all correct processors have at least $n - t$ identical values in Ps and stop. This contradicts the assumption that there is a processor not convinced by round $f + 2$. □

LEMMA 4.9. *All correct processors have the same value stored in s by round $t + 1$.*

PROOF. Recall that all processors stop by round $t + 1$ and give their outputs by round $t + 2$. If any correct processor is convinced by round $t + 1$, then by Lemmas 4.3 and 4.4, all correct processors output the same value by round $t + 2$. Since each correct processor gives as output the value it has stored in s in the round in which it stops, in this case, all correct processors must have stored the same (final) value in s by round $t + 1$.

Assume not all correct processors have the same value stored in s by round $t + 1$. Then no correct processor is convinced by round $t + 1$, and, by Lemma 4.4, no value is persistent by round t. By Lemma 4.7, there is at most one fault not discovered by all correct processors by the end of round t. Also no correct processor stopped by round t. At round t, the one undiscovered fault must separate a witness set to prevent g correct processors from storing the same value in s, for otherwise that value would be persistent at round t. By Lemma 4.6, this undiscovered fault is put in X in round $t + 1$ by every correct processor. Since each correct processor defaults values corresponding to all faulty processors to 0, all correct processors agree on Ps and hence s at the end of round $t + 1$. This contradiction of our original assumption completes the proof. □

If $n > \max(4t, 2(t + (t-1)^2))$, then using EAGREE the correct processors reach eventual agreement by Lemma 4.9 (condition (i)) and Lemma 4.5 (condition (ii)). By Lemma 4.8 and its specification EAGREE requires at most $\min(f+2, t+1)$ rounds of information exchange. This completes the proof of Theorem 4.1. □

5. *Open Problems*

Several unauthenticated deterministic EBA algorithms are known; but none attains the lower bounds of Sections 2 and 3 for all n and t with $n > 3t$ [1, 8, 20, 23]. The question even remains open for authenticated algorithms: Is there a deterministic EBA algorithm that attains the lower bounds for all n and t with $n > 3t$ when the faults are restricted not to corrupt a given authentication protocol? When the faults are restricted to crash, however, the lower bounds are known to be attainable: Fischer and Lamport provide a simple algorithm for EBA that achieves early stopping by round $f+2$ (M. Fischer and L. Lamport, private communications).

ACKNOWLEDGMENTS. The authors would like to thank Amotz Bar-Noy and the referees for helpful comments on earlier versions of this work.

REFERENCES

1. COAN, B. A communication-efficient canonical form for fault-tolerant distributed protocols. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing* (Calgary, Canada). ACM, New York, 1986, pp. 63–72.
2. CRISTIAN, F., AGHILI, H., STRONG, R., AND DOLEV, D. Atomic broadcast: from simple message diffusion to Byzantine agreement. In *Proceedings of of the 15th International Conference on Fault Tolerant Computing* (June). 1985, pp. 1–7.
3. DEMILLO, R. A., LYNCH, N. A., AND MERRIT, M. J. Cryptographic protocols. In *Proceedings of the 14th ACM SIGACT Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, 1982, pp. 383–400.
4. DOLEV, D. Unanimity in an unknown and unreliable environment. In *Proceedings of the 22nd IEEE Annual Symposium on Foundations of Computer Science.* IEEE, New York, 1981, pp. 159–168.
5. DOLEV, D. The Byzantine generals strike again. *J. Algor. 3* (1982), 14–30.
6. DOLEV, D., FISCHER, M., FOWLER, R., LYNCH, N., AND STRONG, R. An efficient algorithm for Byzantine agreement without authentication. *Inf. Control 3* (1983), 257–274.
7. DOLEV, D., AND REISCHUK, R. Bounds on information exchange for Byzantine agreement. *J. ACM 32*, 1 (Jan. 1985), 191–204.
8. DOLEV, D., RIESCHUK, R., AND STRONG, R. 'Eventual' is earlier than 'Immediate'. In *Proceedings of the 23rd IEEE Annual Symposium on Foundations of Computer Science.* IEEE, New York, 1982, pp. 196–203.
9. DOLEV, D., AND STRONG, H. R. Polynomial algorithms for multiple processor agreement. In *Proceedings of the 14th ACM SIGACT Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, 1982, pp. 401–407.
10. DOLEV, D., AND STRONG, H. R. Distributed commit with bounded waiting. In *Proceedings of the 2nd Symposium on Reliability in Distributed Software and Database Systems* (Pittsburgh, Pa., July). 1982, pp. 53–60.
11. DOLEV, D., AND STRONG, H. R. Requirements for agreement in a distributed system. In *Proceedings of the 2nd International Symposium on Distributed Data Bases* (Sept.). 1982, pp. 115–129.
12. DOLEV, D., AND STRONG, H. R. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput. 12* (1983), 656–666.
13. DWORK, C., AND MOSES, Y. Knowledge and common knowledge in a Byzantine environment I: Crash failures. In J. Halpern, ed., *Theoretical Aspects of Reasoning about Knowledge* (Monterey, Calif., Mar. 19–22). Morgan Kaufman, San Mateo, Calif., 1986, pp. 149–170.
14. FISCHER, M., FOWLER, R., AND LYNCH, N. A simple and efficient Byzantine generals algorithm. In *Proceedings of the 2nd Symposium on Reliability in Distributed Software and Database Systems* (Pittsburgh, Pa., July). 1982, pp. 46–52.

15. FISCHER, M., AND LYNCH, N. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett. 14* (1982), 183–186.
16. LAMPORT, L. The weak Byzantine generals problem. *J. ACM 30*, 4 (Oct. 1983), 668–676.
17. LAMPORT, L. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Progr. Lang. Syst. 6*, 2 (Apr. 1984), 254–280.
18. LAMPORT, L., AND MELLIAR-SMITH, P. M. Synchronizing clocks in the presence of faults. *J. ACM 32*, 1 (Jan. 1985), 52–78.
19. LAMPORT, L., SHOSTAK, R., AND PEASE, M. The Byzantine generals problem. *ACM Trans. Prog. Lang. Syst. 4*, 3 (July 1982), 382–401.
20. MOSES, Y., AND WAARTS, O. Coordinated traversal: $(t + 1)$-round Byzantine agreement in polynomial time. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*. IEEE, New York, 1988.
21. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreements in the presence of faults. *J. ACM 27*, 2 (Apr. 1980), 228–234.
22. SCHNEIDER, F. Byzantine generals in action: Implementing fail-stop processors. *ACM Trans. Comput. Syst. 2*, 2 (May 1984), 146–154.
23. SRIKANTH, T., AND TOUEG, S. Byzantine agreement made simple: Simulating authentication without signatures. *Dist. Comput. 2* (1987), 80–94.
24. TOUEG, S., PERRY, K., AND SRIKANTH, T. Fast distributed agreement. *SIAM J. Comput. 16* (1987), 445–457.

RECEIVED JUNE 1983; REVISED JANUARY 1987, JANUARY AND NOVEMBER 1988, AND JULY AND NOVEMBER 1989; ACCEPTED NOVEMBER 1989