Fast Distributed Agreement*

Sam Toueg
Kenneth J. Perry
T.K. Srikanth

TR 84-621

July 1984
(Revised December 1984)
(Revised September 1986)

Department of Computer Science
Cornell University
Ithaca, NY 14853

* Partial support for this work was provided by the National Science Foundation under grant MCS-8303135.

Fast Distributed Agreement†

Sam Toueg

Kenneth J. Perry

T.K. Srikanth

Cornell University
Ithaca, New York 14853

Abstract

We describe a Byzantine Agreement algorithm, with early stopping, for systems with arbitrary process failures. The algorithm presented is simpler and more efficient than those previously known. It was derived using a broadcast primitive that provides properties of message authentication and thus restricts the disruptive behavior of faulty processes. This primitive is a general tool for deriving fault-tolerant algorithms in the presence of arbitrary failures.

1. Introduction

Fault-tolerance is one of the main advantages claimed by distributed systems. A common technique is to replicate processors, and reach agreement on inputs and results. For this reason, much attention has been paid recently to algorithms for reaching processor agreement in failure-prone distributed systems.

We consider a collection of n distributed, potentially faulty processes that communicate through a completely connected and reliable network. In this network, the receiver of a message can always identify the immediate sender of this message. The processes execute a synchronous algorithm in which a distinguished process, the *General*, broadcasts a value m, and all the processes attempt to reach agreement on the value broadcast. *Correct* processes faithfully follow the algorithm, while *faulty* processes may deviate from it. The algorithm solves the *Byzantine Agreement* (or *Byzantine Generals*) problem if it satisfies the properties:

† Partial support for this work was provided by the National Science Foundation under grant MCS-8303135.

validity: If the *General* is correct then all correct processes agree on m.

agreement: All correct processes agree on a common value.

This problem is recognized as a fundamental problem of fault-tolerant computing, and has been extensively studied under various types of failures [Peas80, Lamp82a, Lamp82b, Dole82c, Dole83, Hadz83, Perr86].

The most restrictive type of failure considered is *crash* faults, where processes fail by halting. Even under this restricted model of failure, it was shown that at least $t+1$ phases† of message exchange are needed to reach agreement by any algorithm that overcomes up to t faulty processes [Lamp82a].

This lower bound of $t+1$ on the number of phases required to reach Byzantine Agreement has been used to argue that solutions to this problem are too expensive to be practical. However, by relaxing certain requirements of the problem, agreement can be reached in fewer than t phases, if f, the *actual* number of failures that occur during execution of the algorithm, is less than t, the *maximum* number of failures that the algorithm can overcome.

Dolev et al. [Dole82b] identify two types of Byzantine Agreement algorithms, depending on whether the action resulting from the agreement is to be synchronized or not. These are referred to as *immediate* Byzantine Agreement and *eventual* Byzantine Agreement.

Byzantine Agreement is *immediate* if correct processes can also agree on the phase number at which they reach agreement. Thus, once immediate agreement is reached, processes not only agree on the action to be performed, but are also sure that all correct processes perform the action in the same phase. If this condition is relaxed, agreement is *eventual*: we are only guaranteed that if a correct process decides on some value, then every correct process eventually decides on the same value. This type of agreement is sufficient for many applications where simultaneity is not necessary, for example, when committing a transaction in database systems.

---

† Informally, a phase consists of a synchronized message exchange where processes first broadcast messages (according to their state), then wait to receive messages sent by other processes in the same phase, and change their state accordingly.

Eventual Byzantine Agreement can be reached in time proportional to f rather than t. Algorithms with this property are said to exhibit *early stopping* [Dole82a]. With these algorithms, one pays for the fault-tolerance actually needed, rather than always paying the worst-case time complexity.

Agreement algorithms with early stopping have previously been developed for systems with crash faults [Lamp82b], omission faults (where processes fail by halting or by omitting to send some of the required messages) [Hadz83], and *sr*-omission faults (where processes may also fail to receive messages) [Perr86]. Agreement algorithms with early stopping were also derived for systems with *arbitrary* failures [Dole82b, Reis82]. In this model of failure, processes deviate arbitrarily from the algorithm, even maliciously colluding with one another in order to foil agreement (these are also known as *Byzantine* faults).

However, the algorithms proposed in [Dole82b, Reis82] are generally regarded as complicated and difficult to understand. Furthermore, the algorithm in [Reis82] cannot tolerate more than $n/20$ simultaneous failures. The first algorithm in [Dole82b] reaches eventual Byzantine Agreement in $\min(2f+5, 2t+3)$ phases and requires $O(nt^2 f)$ messages. Thus, when t failures occur, this algorithm is less efficient, in time and message complexity, than the one without early stopping in [Srik84]. In [Dole82b] it is also shown that at least $\min(f+2, t+1)$ phases are required to reach eventual Byzantine Agreement in the worst case. The second algorithm in [Dole82b] meets this lower bound, but requires $n > 2t^2+3t+5$ processes.

In this paper, we describe a simple early-stopping algorithm that overcomes up to $n/3$ faulty processes. It terminates in $\min(2f+4, 2t+1)$ phases and requires $O(nt^2)$ messages. Hence, it is simpler, terminates earlier and has a lower communication complexity than the first algorithm in [Dole82b].

The algorithm in this paper was inspired by an early-stopping algorithm that assumes message authentication [Perr84]. To remove this assumption, we used a broadcast primitive that provides properties of authentication. This primitive is interesting and important in its own right: it effectively restricts the disruptive behavior of faulty processes, and hence simplifies the design of fault-tolerant algorithms for systems with arbitrary failures. Several other applications of such primitives that resulted in efficient fault-tolerant algorithms are described in [Srik84, Srik85].

The remainder of this paper is organized as follows. In Section 2, we describe the properties of the broadcast primitive that is needed by our algorithm. Section 3 presents an early-stopping Byzantine Agreement algorithm for binary-valued messages (the multivalued case is considered in Section 6). In Section 4, we give an implementation of the broadcast primitive. The complexity of the algorithm is investigated in Section 5. Optimizations are given in Section 7. A discussion of the results concludes the paper.

2. Properties of the broadcast primitive

[Srik84] introduced a broadcast primitive that provides properties of authenticated broadcasts without using digital signatures. With this primitive, a process p wishing to broadcast m in round k (where a round is a *logical* phase), executes *broadcast*(p,m,k), and correct processes *accept* (p,m,k) in round k or later. Accepting a message in round k corresponds to receiving this message directly from the sender. Accepting a message in a subsequent round corresponds to receiving a message that was relayed by other processes. The broadcast primitive provides the following properties:

1. *Correctness*: If correct process p executes broadcast(p,m,k) in round k, then every correct process accepts (p,m,k) in the same round.
2. *Unforgeability*: If correct process p does not execute broadcast(p,m,k), then no correct process ever accepts (p,m,k).
3. *Relay*: If a correct process accepts (p,m,k) in round r, then every other correct process accepts (p,m,k) in round $r+1$ or earlier.

These are properties of authenticated broadcasts. Property 1 shows that the primitive indeed implements broadcast. A message broadcast by a correct process at the beginning of round k is received by all correct processes at the end of round k.

Property 2 asserts that a faulty process cannot forge the messages of a correct process. Accepting (p,m,k) corresponds to receiving a message m with p's signature in an authenticated system: in both cases, it can be deduced that if p is correct then p is indeed the originator of m.

Property 3 captures the relaying of signed messages: if a process receives a message m with p's signature in round r and relays it to all processes in round $r + 1$, then all processes receive this signed message by round $r + 1$.

The broadcast primitive introduced in [Srik84] achieves the above three properties of authentication without using digital signatures. It requires $n > 3t$ processes, and two phases of communication exchange for each logical round of the algorithm in which it is applied. Replacing signed communication with this primitive automatically translated several *authenticated* algorithms (i.e., algorithms that assumed message authentication using unforgeable digital signatures), into *non-authenticated* ones (i.e., algorithms that do not need this assumption) [Srik84, Srik85].

Similarly, we could directly translate the authenticated Byzantine Agreement algorithm with early-stopping described in [Perr84] into a non-authenticated one. However, since the authenticated algorithm in [Perr84] terminates in $\min(2f+4, 2t+2)$ phases, the non-authenticated translation would terminate in $\min(4f+8, 4t+4)$ phases, and this is too inefficient.

The authenticated algorithm in [Perr84] uses a mechanism that enables a process to detect broadcasts by other processes, even if it does not directly receive the broadcast message. To achieve this "detection of broadcasts" property, the authenticated algorithm required two phases of communication per round. In this paper, we extend the primitive of [Srik84] to achieve this property without any additional cost in the number of phases. Using this modified primitive allows us to derive a non-authenticated early-stopping algorithm that does not require any more phases than the authenticated one in [Perr84].

The "detection of broadcasts" property allows each correct process to maintain a set, *broadcasters*, that contains the names of processes that have broadcast messages using the primitive. The set *broadcasters* is initially empty. The primitive updates *broadcasters* as follows:

4. *Detection of broadcasts*: If a correct process accepts (p,m,k) in round k or later, then every correct process has $p \in$ *broadcasters* at the end of round $k + 1$. Furthermore, if correct process p does not execute *broadcast* then a correct process can never have $p \in$ *broadcasters*.

Informally, if p broadcasts a message in round k, Property 4 provides that the *initiation* of this broadcast is detected by all correct processes by round $k+1$, even if the first correct process to accept p's message does so in a later round.

The extended primitive is described in details in Section 4. In the next section, we present a simple, non-authenticated Byzantine Agreement algorithm with early-stopping that is based on this extended primitive.

3. A binary Byzantine Agreement algorithm with early stopping

3.1. Informal description

In Figure 1, we present a non-authenticated Byzantine Agreement algorithm with early stopping, for systems of $n>3t$ processes.† We first assume that $m \in \{0,1\}$. This *binary* algorithm is later extended to the case where m is drawn from an arbitrary universe **M**.

In our algorithm, 1 is the only value broadcast by correct processes. The value 0 is decided upon only by default. To decide 1 in round r, a correct process p must have accepted r messages, each message originated by a distinct process in a distinct round. It is also required that the originator of the first of these messages be the *General*. If p decides 1 in round r, it broadcasts 1 in round $r+1$ and stops. Properties 1 and 3 of the broadcast primitive ensure that p's message, and all r messages that p accepted by round r, will also be accepted by all correct processes by round $r+1$, so they all decide 1 and stop.

To decide 0 and stop in round r, a process must be sure that no correct process will later decide 1. This is the case if it has proof that in some previous round (say round $i<r$) no process broadcast a message. (Note that to decide 1 after round i, a process must accept a message broadcast in round i.) To have such a proof, each process keeps a list of broadcasters using Property 4. If a process p discovers that there are fewer than $r-1$ broadcasters by round r, this is proof that there were no broadcasters in some previous round i. By Property 4, no correct process *ever* accepts a message originating in round i. Therefore, no process can later decide 1. Hence, p can safely decide 0 and stop.

In Theorem 3.2 we show that this algorithm terminates by round $\min(f+2,\ t+1)$, where f is the number of processes that are actually faulty during execution.

† It is known that the Byzantine Agreement problem can be solved without authentication only if $n>3t$ [Peas80].

```
p: broadcasters := φ;
   if p = General then value = m else value = 0 fi     /* note that m ∈ {0,1} */
   for r = 1 to t + 1 do
      if value ≠ 0 then broadcast (p,1,r); stop fi
      consider broadcasters as updated in round r:
      if | broadcasters | < r − 1 then stop fi
      if in rounds r' ≤ r accepted (p_i,1,i) for all i, 1 ≤ i ≤ r,
         where p_1 is the General, and the p_i's are distinct,
         then value := 1 fi
   od
```

Figure 1. The early-stopping binary Byzantine Agreement algorithm for process p.

3.2. Proof of correctness

We begin with some definitions. A process stops *at round* k if $r = k$ when it stops. A process *decides* m *at round* k if it stops at round k with $value = m$.

Lemma 3.1: All correct processes decide a value after at most $t+1$ rounds.

Proof: The algorithm consists of a loop that terminates after at most $t + 1$ rounds. □

We now show the algorithm's *validity*.

Lemma 3.2: If the *General* is correct and broadcasts m, then all correct processes decide m.

Proof: Since the *General* is correct, it sets $value = m$. Two cases are possible.

1. If $m = 1$, then the *General* broadcasts (*General*, 1, 1) in round $r = 1$, and stops. In round 1, all other correct processes have $|broadcasters| \geq r - 1$ and, by Property 1 of our broadcast primitive, they accept (*General*, 1, 1). Therefore they all set *value* to 1 in round 1, and decide 1 in round 2.
2. If $m = 0$, then the *General* does not broadcast any message in round 1. Therefore, correct processes never accept a (*General*, 1, 1) message. Hence, they never update their *value* variable, initially set to 0, and they all decide 0 when they stop. □

To prove the algorithm's *agreement* property we need the following two lemmas.

Lemma 3.3: If a correct process decides 1 at round r, then every correct process that executes round r (i.e., did not stop in an earlier round) decides 1 at round r or $r + 1$.

Proof: Let p be a correct process that decides 1 at round r, where $1 \leq r \leq t + 1$. Consider when p sets *value* to 1. If it is at the beginning of round 1, then p must be the *General* and the proof of Lemma 3.2 (case 1) can be applied. There are two other possible cases:

1. Process p sets *value* to 1 at the end of round $r - 1$. Therefore, by round $r - 1$, p must have accepted $(p_i, 1, i)$ messages for all i, $1 \leq i \leq r - 1$, where p_1 is the *General*, and the p_i's are distinct. Note that $p \neq p_i$ for all i, $1 \leq i \leq r - 1$. In round r, p broadcasts $(p, 1, r)$. From Property 4 of our broadcast primitive, since p accepted $(p_i, 1, i)$ for all i, $1 \leq i \leq r - 1$, then every correct process that accepts round r messages must have $p_i \in$ *broadcasters* for all i, $1 \leq i \leq r - 1$, after accepting round r messages. So at this point every correct process has $|$ *broadcasters* $| \geq r - 1$, and therefore cannot stop and decide 0 in round r.

   By Properties 1 and 3 of our broadcast primitive, all correct processes that do not stop in round r must accept $(p, 1, r)$ and $(p_i, 1, i)$ for all i, $1 \leq i \leq r - 1$, by the end of round r. Hence, they all set *value* to 1 in round r, and decide 1 in round r or $r + 1$.

2. Process p sets *value* to 1 at the end of round r. Since p also stops in round r it must be that $r = t + 1$. So p must have accepted some messages $(p_i, 1, i)$ for $1 \leq i \leq t + 1$, with distinct p_i's. One of these $t+1$ processes, say p_j, must be correct. By Property 2 of our broadcast primitive, p_j broadcast $(p_j, 1, j)$ in round j, decided 1 in round j, and set *value* to 1 in round $j - 1 < j$. The proof now follows by applying case 1 to p_j. □

From Lemma 3.3 it is clear that if a correct process decides 1 at round r, then no correct process decides 0 at round $r' \geq r$. We now show the analogous result if a correct process decides 0.

Lemma 3.4: If a correct process decides 0 at round r then no correct process decides 1 at round $r' \geq r$.

Proof: Let p be a correct process that decides 0 at round r. By Lemma 3.3, no correct process decides 1 at round $r' = r$. Assume, for contradiction, a correct process q decides 1 at round $r' > r$. We must have $r < t + 1$, and therefore p decided 0 by stopping in round r with $|\mathit{broadcasters}| < r - 1$. Since q decides 1 at round $r' > r$ it must have accepted $(p_i, 1, i)$ messages for all i, $1 \leq i \leq r' - 1$. Hence, by Property 4 of our broadcast primitive, p must have $p_i \in \mathit{broadcasters}$ for all i, $1 \leq i \leq r - 1$, just after accepting round r messages. So p has $|\mathit{broadcasters}| \geq r - 1$ at round r, a contradiction. □

We now prove the algorithm's *agreement* property.

Lemma 3.5: All correct processes decide the same value.

Proof: Let r be the first round in which a correct process decides, and p a correct process that decides in round r. No correct process decides in a round $r' < r$. If p decides 1 then, by Lemma 3.3, no correct process decides 0 in round $r' \geq r$. If p decides 0 then, by Lemma 3.4, no correct process decides 1 in round $r' \geq r$. Note that all correct processes stop, and therefore decide. □

Theorem 3.1: The algorithm in Figure 1 solves the Byzantine Agreement problem.

Proof: Immediate from Lemmas 3.1, 3.2 and 3.5. □

We now show the algorithm is early-stopping.

Theorem 3.2: If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop by round $\rho = \min(f + 2, t + 1)$. †

Proof: By Lemma 3.1, all correct processes stop and therefore decide by round $t + 1$. Let r be the first round a correct process decides, and p a correct process that decides in round r. There are two possible cases.

1. Suppose p decides 1. If $r = 1$ then p must be the *General*, and the proof of Lemma 3.2 shows that all other correct processes decide 1 at round 2. We now assume $r > 1$. Since p decides 1 at round $r > 1$, then p must have accepted $(p_i, 1, i)$ messages for $1 \leq i \leq r - 1$. Since r is the first round a correct process broadcasts a message, then all p_i's, $1 \leq i \leq r - 1$, must be

† Note that our implementation of the broadcast primitive requires two phases of communication per round.

faulty, and therefore $r - 1 \le f$. From Lemmas 3.3 and 3.4, every correct process decides 1 by round $r + 1 \le f + 2$.

2. If p decides 0, then, by Theorem 3.1, all correct processes also decide 0. Therefore, no correct process ever broadcasts a message. By Property 4 of our broadcast primitive, the variable *broadcasters* of a correct process can contain only names of faulty processes. Hence $|\mathit{broadcasters}| < f + 1$, and any correct process that executes round $f + 2$ must stop at this round. □

Suppose the *General* is correct. From the proof above, if $m = 1$ then all correct processes decide 1 in round 2. However, if $m = 0$ and there are f faulty processes during the execution of the algorithm, it could take $f+2$ rounds for the correct processes to decide 0. This undesirable characteristic is easily eliminated by allowing only the *General* to be inserted in *broadcasters* in round 2. With this modification, if $m = 0$, then the correct processes have $|\mathit{broadcasters}| = 0 < r - 1$ in round 2, and stop.

In the next section we describe the broadcast primitive invoked by our Byzantine Agreement algorithm. We postpone the algorithm's complexity analysis to the subsequent section.

4. The broadcast primitive and its communication complexity

The broadcast primitive in [Srik84] provides the three properties of authentication needed by our early-stopping algorithm. In this section, we extend a version of this primitive to provide the fourth property needed by our algorithm: *detection of broadcasts*. This extension is achieved by inserting one statement that updates the variable *broadcasters*.

The resulting primitive is presented in Figure 2. It can be used in systems where $n > 3t$, and simulates the authenticated broadcast of message m by process p in round k of a synchronized algorithm. As presented here, it can be applied to algorithms in which correct processes broadcast at most one message throughout the algorithm (as in the algorithm of Figure 1). The primitive requires two phases of communication exchange for each round of the algorithm in which it is applied: round k corresponds to phases $2k-1$ and $2k$ of the primitive.

There are four types of messages. A process wishing to broadcast sends an *init* message to all processes in the first phase of the broadcast. Processes send *echo* message in the second phase, *init'* in the third phase, and *echo'* in later phases of the broadcast, according to the algorithm described in Figure 2. When a process receives sufficient echoes of a message it *accepts* the message. In the description of the primitive, message fields marked by an "_" (underscore) can contain arbitrary values. For example, $(\_,p,\_,\_)$ refers to any message sent by process p, and $(echo,p,\_,\_)$ refers to any message of type *echo* sent by p.

Lemma 4.1: If a correct process ever sends $(echo',p,m,k)$, then at least one correct process must have sent $(echo',p,m,k)$ in phase $2k+2$.

Proof: Let l be the earliest phase in which any correct process q sends $(echo',p,m,k)$. If $l \geq 2k+3$, process q must have received $(echo',p,m,k)$ messages from at least $n-2t$ distinct processes, i.e., it must have received $(echo',p,m,k)$ from at least one correct process in phase $l-1$ or earlier. Hence, some correct process sends $(echo',p,m,k)$ before phase l, a contradiction. Therefore, $l=2k+2$.□

Lemma 4.2: If a correct process ever sends $(echo',p,m,k)$, then p must have sent $(init,p,m,k)$ to at least one correct process in phase $2k-1$.

Proof: By Lemma 4.1, if a correct process ever sends $(echo',p,m,k)$, then some correct process q must have sent $(echo',p,m,k)$ in phase $2k+2$. Therefore, process q must have received $(init',p,m,k)$ from at least $n-t$ processes in phase $2k+1$. At least $n-2t$ of these processes are correct, and each of them must have received at least $n-2t$ $(echo,p,m,k)$ messages in phase $2k$. Hence, at least one correct process must have sent $(echo,p,m,k)$ in phase $2k$ and it must have received $(init,p,m,k)$ from p in phase $2k-1$.

Theorem 4.1: The broadcast primitive in Figure 2 has the four properties stated in Section 2.

Proof:

Correctness: Since p is correct, every correct process receives $(init,p,m,k)$ in phase $2k-1$ and sends $(echo,p,m,k)$ in phase $2k$. Hence, every process receives $(echo,p,m,k)$ from at least $n-t$ correct processes in phase $2k$ and every correct process accepts (p,m,k) at the end of phase $2k$, i.e. at the end of round k.

Rules for broadcasting and accepting (p,m,k):

Round k:

Phase $2k-1$: processor p sends $(init,p,m,k)$ to all processors.

Each processor executes the following for each $m \in \mathbf{M}$:

Phase $2k$:

if received $(init,p,m,k)$ from p in phase $2k-1$
and received only one $(init,p,\_,\_)$ message in all previous phases
then send $(echo,p,m,k)$ to all;
if received $(echo,p,m,k)$ from $\geq n-t$ distinct processors in phase $2k$
then *accept* (p,m,k);

Round $k+1$:

Phase $2k+1$:

if received $(echo,p,m,k)$ from $\geq n-2t$ distinct processors q in phase $2k$
and received only one $(echo,p,\_,\_)$ message from each such q
then send $(init',p,m,k)$ to all;
if received $(init',p,m,k)$ from $\geq n-2t$ distinct processes in phase $2k+1$
then $broadcasters := broadcasters \cup \{p\}$;

Phase $2k+2$:

if received $(init',p,m,k)$ from $\geq n-t$ distinct processors in phase $2k+1$
then send $(echo',p,m,k)$ to all;
if received $(echo',p,m,k)$ from $\geq n-t$ distinct processors in phase $2k+2$
then *accept* (p,m,k);

Round $r \geq k+2$:

Phase $2r-1,2r$:

if received $(echo',p,m,k)$ from $\geq n-2t$ distinct processors in previous phases
and not sent $(echo',p,m,k)$
then send $(echo',p,m,k)$ to all;
if received $(echo',p,m,k)$ from $\geq n-t$ distinct processors in previous phases
then *accept* (p,m,k);

Figure 2. The broadcast primitive.

Unforgeability: If p is correct and does not execute broadcast(p,m,k), it does not send any message $(init,p,m,k)$ in phase $2k-1$, and no correct process sends $(echo,p,m,k)$ in phase $2k$. Hence, no correct process can accept (p,m,k) in phase $2k$. If some correct process accepts (p,m,k) at the end of phase $2k+2$ or later, it must have received $(echo',p,m,k)$ messages from at least $n-t$ distinct processes, i.e., it must have received $(echo',p,m,k)$ from at least $n-2t$ correct processes. By Lemma 4.2, p must have sent $(init,p,m,k)$ to at least one correct process in phase $2k-1$, a contradiction. Thus, no correct process ever accepts (p,m,k).

Relay: If $r=k$, then q must have received $(echo,p,m,k)$ from at least $n-t$ distinct processes in phase $2k$. Hence, in the same phase, every correct process receives $(echo,p,m,k)$ from at least $n-2t$ distinct processes and sends $(init',p,m,k)$ in phase $2k+1$. Therefore, every correct process sends $(echo',p,m,k)$ during phase $2k+2$ and every correct process accepts (p,m,k) at the end of phase $2k+2$, i.e., round $r+1$. Now, suppose $r \geq k+1$. Let q be a process that accepts (p,m,k) in phase i where $i=2r-1$ or $2r$. Process q must have received $(echo',p,m,k)$ from at least $n-t$ distinct processes by phase i. Hence, every correct process receives $(echo',p,m,k)$ from at least $n-2t$ distinct processes by phase i, and therefore sends $(echo',p,m,k)$ at or before phase $i+1$. Thus, every correct process receives $(echo',p,m,k)$ from at least $n-t$ distinct processes by phase $i+1$, and therefore accepts (p,m,k) by the end of phase $i+1$, i.e., in round $r+1$ or earlier.

Detection of broadcasts: We first show the second part this property. If a correct process adds p to *broadcasters*, then it received $(init',p,\_,\_)$ messages from at least $n-2t$ distinct processes in phase $2k+1$. Therefore, at least one correct process sent an $(init',p,\_,\_)$ message in phase $2k+1$, and it must have received $(echo,p,\_,\_)$ messages from at least $n-2t$ distinct processes in phase $2k$. Hence, at least one correct process must have received an $(init,p,\_,\_)$ message directly from p in phase $2k-1$. Therefore p executed the broadcast primitive.

We now prove the first part of Property 4. If a correct process accepts (p,m,k) in phase $2k$ (round k), then it must have received $(echo,p,m,k)$ messages from at least $n-t$ distinct processes in phase $2k$. Therefore, every correct process received $(echo,p,m,k)$ messages from at least $n-2t$ distinct correct processes in phase $2k$, and these must be the first $(echo,p,\_,\_)$ message that each one sent. Hence, every correct process sends $(init',p,m,k)$ messages in phase $2k+1$. So, every correct process receives $(init',p,m,k)$ messages from at least $n-t$ distinct processes in phase $2k+1$, and adds p to *broadcasters* in this phase, i.e., in round $k+1$.

If a correct process accepts (p,m,k) in phase $k' \geq 2k+2$, then it must have received $(echo',p,m,k)$ messages from at least $n-t$ distinct processes by phase k'. Let l be the earliest phase in which any correct process q sends an $(echo',p,m,k)$ message. By Lemmas 4.1 and 4.2, $l = 2k+2$, and q must have received $(init',p,m,k)$ messages from at least $n-t$ distinct processes in phase $2k+1$. Therefore all correct processes received $(init',p,m,k)$ messages from at least $n-2t$ distinct correct processes in phase $2k+1$, and they all add p to *broadcasters* in this phase. □

Note that the broadcast primitive need not terminate if the broadcaster is faulty, i.e., there is no a priori bound on the number of phases needed to establish the four properties. However, when a process stops in round r of an agreement algorithm, then it also stops its execution of the underlying broadcast primitive at the end of round r, i.e., at the end of phase $2r$. We now informally show that processes stopping their participation in the underlying broadcast primitive in the same round they decide does not affect the Byzantine Agreement algorithm presented in the previous section.

Let r be the first round in which a correct process decides, and let p be a process that decides in round r. If p stops and decides 1 in round r then it continues its execution of the broadcast primitive till the end of round r. From the proof of Lemma 3.3, every correct process will set $value = 1$ by the end of round r, and therefore decide 1 when it stops, regardless of p's participation in execution of the broadcast primitive after round r. If p decides 0, stopping its participation in the broadcast primitive can only prevent a message from being accepted by some process, or prevent the insertion of a process's name into the *broadcasters* variable of some other process (so the size of $|broadcasters|$ can only *decrease* compared to its size if p never stops executing the primitive). This cannot cause any correct process to decide 1 or to continue executing the algorithm for more rounds than if p never stopped its execution of the broadcast primitive.

5. The complexity of the binary Byzantine Agreement algorithm

In computing the message complexity of the primitive, we consider only messages sent by correct processes. It is not possible to restrict the number of messages sent by faulty processes.

Lemma 5.1: In an algorithm in which the primitive of Figure 2 is applied, each correct process sends a total of $O(n)$ $(\_,p,\_,\_)$ messages for each process p.

Proof: Consider a given process p. Each correct process p_i accepts at most one $(init,p,\_,\_)$ message from p and thus sends at most one $(echo,p,\_,\_)$ message to each process. p_i considers at most one $(echo,p,\_,\_)$ message from each other process p_j throughout the algorithm. Since p_i sends $(init',p,m,k)$ only on receiving at least $n-2t$ $(echo,p,m,k)$ messages, and since $n>3t$, each correct process sends at most $\lfloor n/(n-2t) \rfloor \leq 2$ $(init',p,\_,\_)$ messages.

We now show that at most 3 distinct $(echo',p,\_,\_)$ messages are sent (to all) by the set of correct processes (and hence by each individual correct process). Let C be a set of $n-t$ correct processes. By Lemma 4.1, if a correct process sends $(echo',p,m,k)$ in any phase, then some correct process q must have sent $(echo',p,m,k)$ in phase $2k+2$. For correct process q to send $(echo',p,m,k)$ in phase $2k+2$, it must receive $(init',p,m,k)$ messages from at least $n-t$ processes, i.e., from at least $n-2t$ correct processes in C. Therefore, the sending of an $(echo',p,m,k)$ by a correct process requires the sending of $(init',p,m,k)$ by $n-2t$ correct processes in C to all processes, i.e., a total of $(n-2t)n$ such messages sent by processes in C. Since each correct process sends at most 2 $(init',p,\_,\_)$ messages to all processes, then processes in C send a total of at most $2(n-t)n$ $(init',p,\_,\_)$. Hence, the total number of distinct $(echo',p,\_,\_)$ sent by the set of correct processes is bounded by $\lfloor 2(n-t)n/(n-2t)n \rfloor \leq 3$. Thus, each correct process sends at most 3 $(echo',p,\_,\_)$ messages. Therefore, each correct process sends at most 6 $(\_,p,\_,\_)$ messages to all processes, i.e., a total of $O(n)$ $(\_,p,\_,\_)$ messages. □

We now analyse the message complexity of the Byzantine Agreement algorithm in Figure 1. By Lemma 5.1, each correct process sends $O(n)$ $(\_,p,\_,\_)$ messages throughout the algorithm for each process p. There are n processes, and therefore each correct process sends $O(n^2)$ messages throughout the algorithm. Hence, all correct processes send a total of $O(n^3)$ messages.

However, we can apply some standard techniques [Dole83, Srik84] to reduce the number of messages to $O(nt^2)$ as follows. A set of $3t+1$ *active* processes execute the algorithm as described in Figure 1. Broadcasts by the active processes are to *all* processes. The rest of the processes are *passive*, they only receives messages sent by the active processes. By Theorem 3.2, the active processes decide the same value $m \in \{0,1\}$ by round $\rho = \min(f + 2, t + 1)$. It is easy to see that the passive processes can also be made to decide m by round ρ, from the messages they received from active processes.

Theorem 5.1: The binary Byzantine Agreement algorithm described above requires $O(nt^2)$ messages and a total of $O(nt^2 \log n)$ message bits. If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop within $\min(2f + 4, 2t + 2)$ phases.

Proof: Only active processes p broadcast, and therefore each correct active process sends $(\_,p,\_,\_)$ messages only for active p's. Hence, by Lemma 5.1, each correct active process sends at most $O(nt)$ messages during the execution of the algorithm. So all correct active processes send a total of $O(nt^2)$ messages. We assume it takes $\log n$ bits to encode the name of a process. By Theorem 3.2, the algorithm terminates within min $(f + 2, t + 1)$ rounds. Since the broadcast primitive requires two phases of communication for each round, the algorithm terminates within $\min(2f + 4, 2t + 2)$ phases. □

6. A multivalued Byzantine Agreement algorithm with early stopping

In the Appendix, we extend our algorithm to the case where the *General*'s value m is drawn from an arbitrary universe $\mathbf{M}$. In that section we prove the following theorem.

Theorem 6.1: The multivalued Byzantine Agreement algorithm requires $O(nt^2)$ messages and a total of $O(nt \cdot (\log n + \log |\mathbf{M}|) + nt^2 \cdot \log n)$ message bits. If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop within $\min(2f + 5, 2t + 3)$ phases.

Proof: In the Appendix. □

7. Optimizations

We can easily modify the two Byzantine Agreement algorithms to terminate one phase earlier without increasing the communication complexity. Note that a process broadcasting a message in round $t+1$, the last round, can send the message *directly* to all the processes (without using the broadcast primitive) the reason being that messages sent in the last round are not relayed further, and therefore they do not need to be authenticated. So the last round of the algorithms takes only one communication phase instead of two, and termination occurs at phase $\min(2f + 4, 2t + 1)$ and $\min(2f + 5, 2t + 2)$ for the binary and multivalued algorithms, respectively. For this to work however, a slightly modified version of the broadcast primitive must be used in round t, to ensure that a message accepted by a correct process in round t is accepted by all correct processes just one phase after round t [Srik84].

The concept of a process *committing* a value v during the execution of a Byzantine Agreement algorithm was introduced in [Dole82b]. Informally, a process commits a value v if it knows that all correct processes will decide v by the time the algorithm terminates. This can happen before the process is able to stop its participation in the algorithm. However, as soon as a process commits a value, it can use it for the purpose for which the algorithm was started. With our algorithms, it is not difficult to see that processes may commit one phase earlier than they stop, without increasing the communication complexity. Processes can commit at phase $\min(2f + 3, 2t + 1)$ and $\min(2f + 4, 2t + 2)$ for the binary and multivalued algorithms, respectively.

Finally, the communication complexity can be further reduced to $O(nt \cdot \log t + t^3 \cdot \log t)$ bits and $O(nt \cdot (\log t + \log |\mathbf{M}|) + t^3 \cdot \log t)$ bits for all the binary and multivalued algorithms above, at the cost of one additional phase. This is achieved by executing the algorithm among active processes only. At termination, the value decided is directly broadcast to the passive processes and they decide on the majority of the values received.

8. Discussion and conclusions

Non-authenticated algorithms for reaching agreement in systems with arbitrary failures have been notoriously complicated, and difficult to derive and prove correct. Such algorithms are even more sophisticated and difficult to understand when they exhibit early stopping.

In this paper, we took advantage of the properties provided by our broadcast primitive to derive a conceptually simple early-stopping algorithm. The algorithm derived is also more efficient than those previously known. It is interesting to note that the communication complexity of the best known Byzantine Agreement algorithm *without* early stopping is no lower than ours. This is somewhat surprising, since one would expect to pay some price in communication complexity for the property of stopping early.

The algorithm was derived and proved correct assuming four properties of broadcasts. The algorithm (and its proof) is independent of any particular implementation of the broadcast primitive that achieves these four properties. There may be more efficient or simpler implementations of such a primitive than the one we proposed.

The results presented in this paper illustrate once more the power and versatility of our primitive. It provides properties of message authentication and other properties that effectively restrict the disruptive behavior of faulty processes. The problem of designing algorithms for systems with arbitrary failures is thereby simplified. We have applied the same technique to problems other than agreement, e.g., to the problem of optimal clock synchronization in the presence of arbitrary faults [Srik85, Beck86].

Appendix

The multivalued algorithm

We consider the case where the *General*'s value m is drawn from an arbitrary universe **M**. In this case, the *General* executes a modified broadcast primitive in the first round. This primitive is presented in Figure 4, and has the following properties:

Theorem A.1: The broadcast primitive in Figure 4 has the four properties described in Section 2, and the following *uniqueness* property:

5. If a correct process accepts $(p,m,1)$, then no correct process ever accepts $(p,m',1)$ with $m' \neq m$.

Proof: The proof of the first four properties is similar to the proof of the four properties in Theorem 3.2, and is left to the reader. We now sketch the proof of Property 5.

We first note that if a correct process sends $(echo',p,m,1)$ in Phase 2, then no correct process sends $(echo',p,m',1)$ in Phase 2 for any $m' \neq m$. Otherwise, at least $n-t$ processes sent $(echo,p,m,1)$ and $n-t$ processes sent $(echo,p,m',1)$ in Phase 1. Since $n>3t$, this implies that at least one correct process sent both $(echo,p,m,1)$ and $(echo,p,m',1)$ in Phase 1, and this is not possible.

We also note that if a correct process accepts $(p,m,1)$, then at least one correct process sends $(echo',p,m,1)$, and therefore at least one correct process sends $(echo',p,m,1)$ in Phase 2. Property 5 is an immediate consequence of the two remarks above. □

This primitive is denoted *monobroadcast* because of the *uniqueness* property. Note that the first round takes 3 phases, but the later rounds take two phases each. Consider the communication complexity of *monobroadcast*:

Lemma A.1: In an algorithm in which the *General* invokes *monobroadcast* in the first round, each correct process sends a total of $O(n)$ $(\_,General,\_,1)$ messages.

Proof: The proof is similar to the proof of Lemma 5.1, and will be omitted here. □

The multivalued Byzantine Agreement algorithm is derived from the binary one by having the *General* execute *monobroadcast* to broadcast $m \neq 0$ in round 1. All broadcasts initiated in later rounds use *broadcast*. A process sets its value to $m \neq 0$ in round r if it accepted $(General,m,1)$ and $(p_i,1,i)$ for

Rules for broadcasting and accepting $(p,m,1)$ for $p = General$:

Round 1:

Phase 0 : processor p sends $(init,p,m,1)$ to all processors.

Each processor executes the following for each $m \in \mathbf{M}$:

Phase 1:

if received $(init,p,m,1)$ from p in phase 0
and received only one $(init,p,\_,\_)$ message in this phase
then send $(echo,p,m,1)$ to all;
if received $(echo,p,m,1)$ from $\geq n-2t$ distinct processes in phase 1
then $broadcasters := broadcasters \cup \{p\}$;

Phase 2:

if received $(echo,p,m,1)$ from $\geq n-t$ distinct processors q in phase 1
and received only one $(echo,p,\_,\_)$ message from each such q
then send $(echo',p,m,1)$ to all;
if received $(echo',p,m,1)$ from $\geq n-t$ distinct processes in phase 2
then $accept\ (p,m,1)$;

Round $r \geq 2$:

Phase $2r-1,2r$:

if received $(echo',p,m,1)$ from $\geq n-2t$ distinct processors in previous phases
and not sent $(echo',p,m,1)$
then send $(echo',p,m,1)$ to all;
if received $(echo',p,m,1)$ from $\geq n-t$ distinct processors in previous phases
then $accept\ (p,m,1)$;

Figure 4. The monobroadcast primitive invoked by the *General* in the first round.

all i, $2 \leq i \leq r$, where the p_i's are distinct.

Theorem A.2: The algorithm described above is a multivalued Byzantine Agreement algorithm. If there are f faulty processes during the execution of the algorithm, then all correct processes decide by round $\rho = \min(f+2, t+1)$.

Proof: The *uniqueness* property of the *General*'s broadcast primitive effectively reduces the multivalued case to the binary case, and the proof of Theorem A.2 is nearly identical to the proofs of Theorems 3.1 and 3.2. It will not be repeated here. □

To reduce the message complexity, the algorithm could be executed on a selected set of $3t+1$ of active processes, as explained in Section 5. The complexity of the resulting multivalued Byzantine Agreement algorithm is as follows.

Theorem A.3: The multivalued Byzantine Agreement algorithm requires $O(nt^2)$ messages and $O(nt\cdot(\log n+\log|\mathbf{M}|) + nt^2\cdot\log n)$ message bits. If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop within $\min(2f + 5, 2t + 3)$ phases.

Proof: By Lemma A.1, each correct active process sends $O(n)$ (_,*General*,_,1) messages. Hence, all active processes send a total of $O(nt)$ (_,*General*,_,1) messages. Assuming it takes $\log n$ bits to encode a process's name, and $\log|\mathbf{M}|$ to encode a message $m \in \mathbf{M}$, then each (_,*General*,m,1) message is $\log n+\log|\mathbf{M}|$ bits. By Lemma 5.1, the total number of other messages sent by the active processes is $O(nt^2)$, and each such message is $O(\log n)$ bits. For the number of phases required for termination, note that the first round takes three phases, and all subsequent ones take two phases, and apply Theorem A.2. □

References

Beck86 M. Beck, T.K. Srikanth, S. Toueg, Implementation issues in clock synchronization. To appear in *Lecture Notes of the Asilomar Workshop on Fault-Tolerant Distributed Computing*, March 16, Asilomar, California, 1986, Springer-Verlag.

Dole81 D. Dolev, Unanimity in an unknown and unreliable environment, *Proceedings 22nd Annual Symposium on Foundations of Computer Science*, Nashville, Tennessee, pp. 159-168, Oct. 1981.

Dole82a D. Dolev and H. R. Strong, Requirements for agreement in a distributed system, Tech. Rep. RJ 3418, IBM Research Laboratory, San Jose, California, March 1982.

Dole82b D. Dolev, R Reischuck, and H. R. Strong, 'Eventual' is earlier than 'immediate', *Proceedings 23rd Symposium on Foundations of Computer Science,* Chicago, Illinois, pp. 196-203, November 1982.

Dole82c D. Dolev, M.J. Fischer, R. Fowler, N.A. Lynch, R. Strong, An efficient algorithm for Byzantine Agreement without authentication, *Information and Control,* vol. 52, no. 3, 1982.

Dole83 D. Dolev and H. R. Strong, Authenticated algorithms for Byzantine Agreement, *SIAM Journal of Computing,* vol. 12, no. 4, pp. 656-666, Nov. 1983.

Hadz83 V. Hadzilacos, Byzantine Agreement under restricted types of failures (not telling the truth is different from telling lies), Tech. Rep. 19-83, Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts, June 1983.

Lamp82a L. Lamport and M. Fischer, Byzantine Generals and Transaction Commit protocols, Opus 62, SRI International, April 1982.

Lamp82b L. Lamport, R. Shostak, and M. Pease, The Byzantine Generals problem, *ACM Transactions on Programming Languages and Systems,* vol. 4, no. 3, pp. 382-401, July 1982.

Lync82 N. A. Lynch, M. J. Fischer, and R. J. Fowler, A simple and efficient Byzantine Generals algorithm, *Proceedings Second IEEE Symposium on Reliability in Distributed Software and Data Base Systems,* Pittsburgh, Pennsylvania, pp. 46-52, 1982.

Peas80 M. Pease, R. Shostak and L. Lamport, Reaching agreement in the presence of faults, *Journal of the ACM,* vol. 27, no. 2, pp. 228-234, April 1980.

Perr84 K. J. Perry and S. Toueg, An authenticated Byzantine Generals algorithm with early stopping, Tech. Rep. 84-620, Department of Computer Science, Cornell University, Ithaca, New York, June 1984.

Perr86 K. J. Perry and S. Toueg, Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering,* vol. SE-12, no. 3, pp. 477-482, March 1986.

Reis82 R. Reischuk, A new solution for the Byzantine Generals problem, Tech. Rep. RJ 3673, IBM Research Laboratory, San Jose, California, Aug. 1982.

Srik84 T. K. Srikanth and S. Toueg, Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Tech. Rep. 84-623, Department of Computer Science, Cornell University, Ithaca, New York, July 1984. To appear in *Distributed Computing*, Springer-Verlag.

Srik85 T. K. Srikanth and S. Toueg, Optimal clock synchronization. To appear in the *Journal of the ACM*.