Theory Comput Syst (2008) 42: 222–238
DOI 10.1007/s00224-007-9006-9

Efficient Algorithms for Anonymous Byzantine Agreement

Michael Okun · Amnon Barak

Published online: 4 July 2007
© Springer Science+Business Media, LLC 2007

Abstract This paper considers the Byzantine agreement problem in a completely connected network of *anonymous* processors. In this network model the processors have no identifiers and can only detect the link through which a message is delivered. We present a polynomial-time agreement algorithm that requires $3\lfloor(n-t)t/(n-2t)\rfloor+4$ rounds, where $n>3t$ is the number of processors and t is the maximal number of faulty processors that the algorithm can tolerate. We also present an early-stopping variant of the algorithm.

Keywords Byzantine agreement · Anonymous distributed algorithms · Anonymous networks · Distributed fault tolerant computing

1 Introduction

The Byzantine Agreement (BA) problem, originally introduced in the papers of Pease, Shostak and Lamport [19, 23], was intuitively described by a scenario about a group of generals in ancient Byzantium. Some of the generals are traitors, however the loyal generals do not know who the traitors are. The generals need to agree on a plan of a forthcoming battle, in which the legions of the loyal generals must attack or retreat together, otherwise they will be defeated. The generals communicate by messengers that orally pass the messages and it is assumed that all the messengers are loyal (honest).

The accepted formalization of this scenario is a point-to-point network of synchronized processors that are required to agree on a common value, in a way that depends on the processors' initial inputs. In this model, the network has a channel between

M. Okun (✉) · A. Barak
School of Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel
e-mail: mush@cs.huji.ac.il

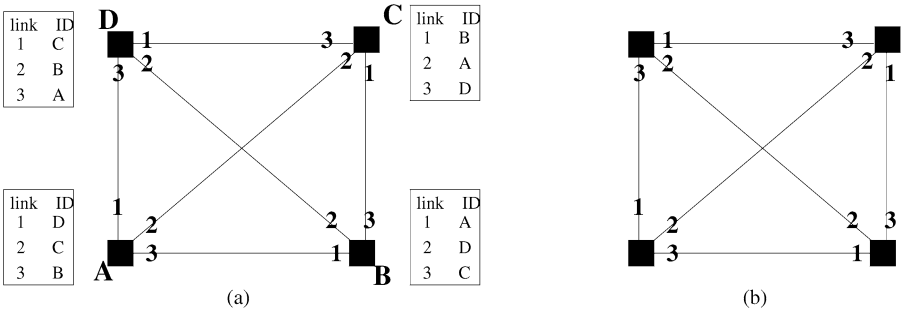

Fig. 1 A system with 4 processors in (**a**) standard model, (**b**) anonymous model

every pair of processors, the processors have unique names (identifiers) that are globally known, and for every received message the identifier of the sending processor is assumed to be known to the receiver. These assumptions mean that each processor has a *mapping* between the numbers of the communication channels (links) and the identifiers of the processors on the other side of the channels (see Fig. 1a). Below we refer to this system model as the "standard model".

This paper shows that some of the assumptions made in the standard model are not required in order to solve the synchronous BA problem. More specifically, we show that no a priori knowledge about the identifiers of other processors is required and that BA can be reached even if no such identifiers exist at all, i.e., the processors are anonymous. In the anonymous case the only information available to the receiver is the identifier of the *link* that delivered the message (see Fig. 1b).

The main result of this paper is a (deterministic) polynomial algorithm that solves the BA problem in a synchronous system of n anonymous processors of which t can be faulty, where $n > 3t$. Obviously this result is optimal in the number of faulty processors, since even in the standard model the agreement problem can not be solved for $n \leq 3t$ [16, 23].

The predominant motivation of this study is to find the minimal conditions which still allow to reach BA. In light of the central role of BA in the area of distributed computing, we believe it is important to understand this aspect of the problem. Finding these conditions also extends the class of problems that are solved by using BA.

The anonymous model is also interesting due to the improved privacy provided to the parties engaged in a BA protocol. Unlike the standard model, in which the parties in the BA protocol reveal the inputs under their real names, the anonymous model only requires that it will be possible to distinguish between messages from different parties. For example, in a BA protocol in the standard model, the messengers sent by a loyal general G_1 who wants to retreat, tell that G_1's opinion is "retreat". Thus G_1 is revealed (under his real name) as a coward (or a traitor). In contrast, in a BA protocol in the anonymous model, the messengers do not reveal (or even know) the name of the general who sent them. The only requirement is that any loyal general G_1 uses the same messenger to communicate with general G_2 during the whole protocol (thus allowing G_2 to distinguish between messages from G_1 and messages from other generals).

1.1 Previous Related Work

BA is fundamental for the design of distributed systems since it provides a basic primitive that allows to reach a coordinated and consistent behavior of the processors. A BA algorithm can also be viewed as a means to simulate a broadcast channel on top of a point-to-point network. Since the BA problem was introduced by Pease, Shostak and Lamport in [19, 23], it is probably the most extensively studied problem in distributed computing. Previous works considered the problem under various timing, topology, authentication and failure assumptions. A general introduction to the subject can be found in [2, 20].

An algorithm for reaching BA in $t + 1$ rounds for $n > 3t$ was presented in [23]. Shortly after, it was shown that no deterministic algorithm can solve the problem in less than $t + 1$ rounds [15]. However, the original BA algorithm requires computation and communication that are exponential in t, thus leaving the design of more efficient algorithms as an open problem. One of the first polynomial BA algorithms was presented in [13]. The algorithm in that paper required $2t + 3$ communication rounds. Following several works which gradually improved the tradeoff between the resilience of the algorithm and its round complexity, Garay and Moses presented in [17] a polynomial algorithm that achieves BA in $t + 1$ rounds for $n > 3t$.

The BA algorithms presented in [12, 13, 24, 26] are of direct relevance to the current paper. These algorithms are based on a *consistent broadcast*[1] primitive [24], which ensures that all the correct processors receive exactly the same messages at almost the same time (see Sect. 3 for a formal definition). A consistent broadcast primitive allows to design simple polynomial algorithms for BA and some other coordination problems (e.g., clock synchronization in presence of Byzantine failures [25]). The algorithms presented in this paper are inspired by this approach.

The study of computations in networks of anonymous processors was initiated in [1], where the connection between anonymous computations and the topological theory of graph coverings was shown. Based on this theory, functions and relations that are computable in anonymous networks were completely characterized in follow up papers, e.g., [6, 7, 28]. In addition to anonymous networks, several anonymous shared memory models were also considered in the past, e.g., in [3]. To the best of our knowledge fault tolerant algorithms for anonymous distributed systems were not studied so far.

Anonymity and privacy issues were also extensively studied in the context of cryptography and secure multi-party computation. One of the main results in this direction shows that it is possible to compute an arbitrary function defined on the inputs of the processors, in a way that leaks no additional information to the faulty processors (unless it is part of the output), for $n > 3t$ [5, 10]. A similar result for a computationally bounded adversary can be achieved for $n > 2t$, by relying on public-key cryptography [18]. An important special case of such a computation is voting, e.g., as in the generals' scenario presented above. Starting from the first papers by Chaum and others [8, 9, 11], electronic voting has become an active research area with significant practical implications.

[1] The name was invented later.

In a recent paper, a question that reminds the one considered in the current work is studied in the context of cryptography-based secure multi-party computation [4]. More specifically, the goal of that paper is to provide a characterization of secure computation in the case of processors that have no established authentication mechanism, such as a deployed public-key infrastructure.

1.2 Organization and Contributions of the Paper

The paper is organized as follows. The next section gives the formal definitions of the anonymous model and the BA problem. Section 3 presents an anonymous BA algorithm for $n > 3t$ that requires $3\lfloor(n-t)t/(n-2t)\rfloor+4$ rounds of computation. An early-stopping version of the algorithm which requires at most $\min(3\lfloor(n-t)t/(n-2t)\rfloor+4,\ 3\lfloor(n-f)f/(n-t-f)\rfloor+3f+9)$ rounds, where f is the actual number of faulty processors in an execution, is presented in Sect. 4. Conclusions and directions for further research are presented in Sect. 5.

2 Definitions

2.1 The System Models

Consider a system with n processors $p_1,\ldots,p_n$, each modeled by a (possibly infinite) state machine. Throughout the paper this kind of numbering is external and is not known to the processors themselves. The processors are arranged in a completely connected network, i.e., there is a dedicated (bidirectional) communication channel (link) between each pair of processors and from each processor to itself. The links connected to a processor p are numbered from 1 to n, where links $1,\ldots,n-1$ are connections to the other processors and link n is a self loop. Each link is modeled as a pair of queues: for incoming messages and for outgoing messages. The state machine that models a processor has a pair of special transitions, called *send* and *receive*, that allow it to place messages in an outgoing queue and receive messages from an incoming queue, respectively. To perform these transitions the number of a queue (in the range $1,\ldots,n$) to which it is applied must be given. In addition, each state machine has a special variable named *input* and a special transition that allows it to get an input from an external source.

Definition In the *anonymous* system model the state machines of all the processors are *identical* (see Fig. 1b).

Definition In the *standard* system model the state machines of different processors may be arbitrarily different.

We note that in a typical distributed algorithm for the standard model the only differences between the state machines of the processors are the processor identifier and the mapping table (see Fig. 1a).

This paper deals with (deterministic) synchronous algorithms in which the processors are assumed to execute in lock-step, which allows to partition the execution into

rounds. Each round consists of two phases: a send phase followed by a receive phase. In the send phase every processor is allowed to place messages in its outgoing queues, and in the receive phase to receive messages from its incoming queues. In both phases a processor can perform internal computations. At the end of each round all the message queues are emptied (regardless of whether they were accessed or not).

Some of the processors may experience arbitrary (Byzantine) failures. After a failure, it is assumed that a processor can generate arbitrary messages. It is convenient to consider the faulty processors as being controlled by a malicious adversary that has a complete information about the execution. A processor that follows the algorithm during the whole execution is said to be correct. Hereafter let t denote the a priori bound on the number of faulty processors, and let f denote the actual number of faulty processors in an execution.

An r round execution $\mathcal{E}$ of a deterministic algorithm in a given network is completely defined by the inputs given to the correct processors and the messages that the faulty processors send to the correct ones. More formally, given a set $\mathcal{C} \subseteq \{1, \ldots, n\}$ of the indexes of the correct processors, a set of inputs to the correct processors $\mathcal{I} = \{(p_i, input_i)\}_{i \in \mathcal{C}}$, and a set of messages sent by the faulty processors to the correct processors $\mathcal{M} = \{(p_i, p_j, k, m)\}_{i \in \{1,\ldots,n\} \setminus \mathcal{C}, j \in \mathcal{C}, r \geq k \geq 1}$ (where the tuple (p_i, p_j, k, m) means that faulty processor p_i sends message m to correct processor p_j in round k), an execution $\mathcal{E}$ is defined as $\mathcal{E} = \mathcal{E}(\mathcal{C}, \mathcal{I}, \mathcal{M})$.

2.2 The Agreement Problem

The Byzantine Agreement (BA) problem [19, 23] requires that the correct processors agree on a common value despite the presence of faulty processors. More formally, each processor gets an input value from a set V and the processors have to output a decision value from V, such that the following conditions are satisfied:

Termination: Every correct processor eventually decides.

Agreement: All the correct processors decide on the same value from V.

Validity: If the input to all the correct processors is $v \in V$, then v is the only possible output value.

The problem stated above is also known as *eventual* BA. In a stronger version, called *simultaneous* BA, all the correct processors are required to decide at exactly the same round [14].

In this paper we consider the *binary* BA problem, in which the set of possible input values to the processors is $\{0, 1\}$. The general BA problem can be solved by executing several instances of the binary agreement algorithm in parallel. It is interesting to note that the algorithm presented in [27] for reducing a general BA problem to a binary BA problem at the cost of two additional rounds, does not use processor names. Thus it works in an anonymous system for $n > 3t$, which provides an additional method for solving the general BA problem using a binary BA algorithm.

3 An Agreement Algorithm with Optimal Resiliency

This section begins with an informal review of the methods used in [12, 13, 24, 26], which are relevant to the current paper. The BA algorithms in these papers are based

on a primitive (called *consistent broadcast* in [20]), that satisfies the following three properties:

(Correctness) If a correct processor p broadcasts a message (p, m, r) in round r then every correct processor *accepts* it by round $r + 1$.
(Unforgeability) If p is correct and does not broadcast (p, m, r) then no correct processor ever accepts (p, m, r).
(Relay) If correct processor accepts (p, m, r) in round r' then every correct processor accepts (p, m, r) in round $r' + 1$ or earlier.

In the authenticated model considered in [12], consistent broadcast is achieved with the help of digital signatures. In the standard model, assuming $n > 3t$ consistent broadcast can be implemented as follows [24]:

(1) To broadcast a message m in round r a processor p sends the message ("*init*", p, m, r) to all the processors.
(2) If a processor receives the message ("*init*", p, m, r) from p in round r, it sends the message ("*echo*", p, m, r) to all the processors.
(3) If a processor receives ("*echo*", p, m, r) messages from at least $n - 2t$ processors, it sends the message ("*echo*", p, m, r) to all the processors, unless it already sent such a message.
(4) Upon receiving ("*echo*", p, m, r) from at least $n - t$ processors, m is accepted.

With the help of a consistent broadcast primitive, a BA algorithm can be constructed in the following manner. The possible decision values (0 and 1) should be considered as candidates in an election held among all the processors. Each processor that decides to support 1 votes in its favor, while refraining from voting is interpreted as supporting 0. A processor votes by broadcasting "1", and it is allowed to do so at most once. The consistent broadcast primitive ensures that the vote of every processor is seen by all the correct processors in the same way and almost at the same time. A correct processor with $input = 1$ must vote right at the beginning of the algorithm. A correct processor with $input = 0$ can possibly vote (in favor of 1) at some later stage of the algorithm, and it does so if it sees that the number of 1's supporters is higher than a certain threshold value. This threshold value increases with time, thus after a certain number of rounds 1 can get no new votes from correct processors with $input = 0$. At this point the algorithm stops and each processor checks how many votes 1 got. If there are at least $2t + 1$ votes, it decides 1, otherwise it decides 0. An additional property of the algorithm, which is crucial for its correctness, is that if at some round a correct processor with $input = 0$ votes, then all the correct processors that did not vote so far do so shortly after.

3.1 Anonymous Byzantine Agreement

Since the definition of the consistent broadcast primitive assumes the existence of unique identifiers for each processor, the approach presented above can not be directly applied to the anonymous model. However, the above voting paradigm, as well as the basic idea of the consistent broadcast protocol ("*echo*" messages that ensure that the correct processors have approximately the same view of the number of votes), can be adapted to the anonymous model.

In the resulting Anonymous BA (ABA) algorithm, a processor votes by sending an *init* message to all the processors, which normally results in an increment of the *proposedCounter* variables of all the correct processors. Each processor is allowed to send *init* messages at most once (a correct processor disregards any additional *init* message received through the same link). In the beginning of every round each correct processor sends the current value of its *proposedCounter* variable to all the processors. This is similar to sending the *echo* messages in the consistent broadcast primitive. Every processor also has a *counter* variable that holds the current number of votes in favor of 1. The increment of the *counter* variable is controlled by the $(t+1)$-th lowest value received in each round. The *proposedCounter* variable might also be updated according to the values received from other processors (this in addition to increments resulting from the receipt of *init* messages): *proposedCounter* is set to the value of the $(2t+1)$-th lowest value received, unless it is lower than its current value. This last operation is intended to provide the counterpart of the Relay property for the anonymous case, since it keeps a lag of at most one round between the values of the *counter* variables between the different correct processors (see also Lemma 3.1 below). The described algorithm solves the ABA problem for $n > 5t$ in $2\lfloor (n-t)t/(n-3t)\rfloor + 2$ rounds. A formal presentation of the algorithm and a proof of its correctness can be found in [22].

We now explain in more detail why the above algorithm works only for $n > 5t$. The basic new problem introduced by the anonymous model is that the t faulty processors can generate *more* than t votes. This follows from the fact that $n-3t$ *init* messages sent by the faulty processors to the correct ones are sufficient for incrementing the *counter* of the correct processors by 1 (since together with the t faulty processors there can be $n-2t$ processors who send the new value, causing the remaining $2t$ correct processors to increase their *proposedCounter* as well, which leads to the increment of the *counter* variables). The total number of links between the correct and faulty processors is $(n-t)t$, so that the number of votes generated by the t faulty processors can be bounded by $\lfloor (n-t)t/(n-3t)\rfloor$. As in the BA algorithms for the standard model, e.g. [24], the number of votes of the correct processors should be higher than twice this number. Informally, the reason is as follows. If initially 1 has at least $\lfloor (n-t)t/(n-3t)\rfloor + 1$ supporters among the correct processors, then all the correct processors vote for 1 right away, so that 1 is guaranteed to have $n-t$ votes. This number of votes must guarantee the election of 1. Otherwise, initially 1 has at most $\lfloor (n-t)t/(n-3t)\rfloor$ supporters among the correct processors, and if no correct processor with $input = 0$ ever decides to vote, then the election result must be 0. Since the faulty processors can generate additional $\lfloor (n-t)t/(n-3t)\rfloor$ votes, we must require $2\lfloor (n-t)t/(n-3t)\rfloor$ votes to be *insufficient* for the election of 1. This explains the $2\lfloor (n-t)t/(n-3t)\rfloor < n-t$ inequality (which is equivalent to $n > 5t$, if we neglect the rounding).

One way to solve the ABA problem for $n > 3t$, is to base the decision of each processor on its own view of the number of *voters*, rather than on the collective number of *votes* (the value of the counter). In such an algorithm, if initially 1 has at least $t+1$ supporters among the correct processors, then all the correct processors vote for 1 right away, so that each correct processors sees $n-t$ voters (i.e., receives *init* messages from at least $n-t$ processors). However, if initially 1 has at most t supporters among the correct processors and no correct processor with $input = 0$ ever

decides to vote, then the result must be 0. Since there are at most t faulty processors, every correct processor sees at most $2t$ voters. These two cases can be distinguished because $2t < n - t$.

The above idea raises a new difficulty. Since the number of votes generated by the t faulty processors can be substantially higher than t, the faulty processors will be able to make the correct processors vote even when all the correct processors start with $input = 0$. To solve this problem, in the new algorithm the first *init* message received from a processor is not ignored only if the number of votes for 1 is already high enough. We note that this verification procedure for the *init* messages is crucial for the construction of an early-stopping ABA algorithm, presented in Sect. 4.

By applying these modifications an ABA algorithm for $n > 3t$ is achieved. The formal presentation of this algorithm can be found in [22]. The running time of this algorithm is $\Theta((n-t)t/(n-3t))$, which is quadratic in the number of processors when $n = 3t + 1$. Intuitively, this is due to the large number of votes that the faulty processors are able to generate. This number can be significantly reduced if the processors use another variable (named *possibleCounter*) to count the *init* messages, *instead* of using *proposedCounter* for that. After this modification the algorithm requires only $O(t)$ rounds, and in the specific case $n = 3t + 1$ it requires $6t + 1$ rounds.

The resulting algorithm is presented in Fig. 2. The formal correctness proof is shown in Sect. 3.2.

3.2 The Correctness Proof

To begin with, observe that only the first *init* message received through a link "is counted", while all the others are ignored (see lines 17, 18 in Fig. 2). This property allows to assume that during the whole execution *every* processor sends *at most one init message* on each of its links. This way the algorithm guarantees that the faulty processors are unable to generate an unbounded number of votes.

For convenience we denote the expression $\lfloor (n-t)t/(n-2t) \rfloor$ by T.

Lemma 3.1 *If by the end of round $r \leq 3T + 3$ the counter of a correct processor p is C, then by the end of round $r + 1$ the counters of all the correct processors are at least C.*

Proof First observe that the value of the counter is updated only in line 15. If by the end of round r the value of the counter of p is $C > 0$ then p must have received (in round r or in a previous round) proposed values that are greater or equal to C from $n - t$ distinct processors. Since at least $n - 2t$ of these messages originated from correct processors, they must have been received by all the processors. According to the algorithm, by the end of round r the value of *proposedCounter* of every correct processor is at least C (see line 14). Thus, the proposed values sent by every correct processor in round $r + 1$ are greater or equal to C. Therefore by the end of that round the counter of every correct processors is at least C. □

Note The property that was shown in Lemma 3.1 corresponds to the Relay property of the consistent broadcast. Next we show that if some correct processor with *input* =

```
INITIAL SETUP:
 1 GET the input value;
 2 sentInits := false;
 3 possibleCounter := proposedCounter := counter := 0;

IN ROUND 1 ONLY:
 4 IF input = 1 THEN
 5    SEND init TO ALL;
 6    sentInits := true;

IN ROUND 1 ≤ r ≤ 3⌊(n − t)t/(n − 2t)⌋ + 4:
 7 SEND ("Po", possibleCounter) AND ("Pr", proposedCounter) TO ALL;
 8 IF counter ≥ t + (r − 1)/3 AND ¬sentInits THEN
 9    SEND init TO ALL;
10    sentInits := true;

11 RECEIVE MESSAGES (sent to the processor in the current round);

12 LET PrV AND PoV be the arrays of the proposed and possible values
   received: PrV[i] and PoV[i] are the proposed and possible values
   received from link i, respectively, or 0 if no such value
   was received from the link;
13 SORT PrV in a decreasing order; SORT PoV in a decreasing order;
14 proposedCounter := max(proposedCounter, PrV[n − 2t], PoV[n − t]);
15 counter := max(counter, PrV[n − t]);
16 j := 0;
17 FOR every link from which init was received for the first time
18    j := j + 1;
19 IF counter ≥ t + (r − 1)/3 OR r = 1 THEN
20    possibleCounter := max(possibleCounter, counter + j);

DECISION RULE TO BE APPLIED AT THE END OF ROUND 3⌊(n − t)t/(n − 2t)⌋ + 4:
21 IF init messages were received from n − t distinct processors THEN
22    DECIDE 1;
23 ELSE
24    DECIDE 0;
```

Fig. 2 Anonymous BA algorithm for $n > 3t$

0 votes in favor of 1, then shortly afterwards all the other correct processors vote in favor of 1.

Lemma 3.2 *If a correct processor p sends init messages in round $3T + 1 \geq r > 1$, then every correct processor sends init messages by round $r + 3$.*

Proof From the algorithm it follows that at the end of round $r - 1$ the counter of p is at least $t + (r - 1)/3$. According to Lemma 3.1, after line 15 in round r the counter of every correct processor becomes at least $t + (r - 1)/3$. Thus the *init* message

sent by p causes every correct processor to set its *possibleCounter* to be at least $t + (r-1)/3 + 1$. This in turn shows that in round $r+1$ the possible value sent by every correct processor is at least $t + (r-1)/3 + 1$, so that by the end of that round the *proposedCounter* of every correct processors is at least $t + (r-1)/3 + 1$. It follows that by the end of round $r+2$ the counter of every correct processor becomes at least $t + (r-1)/3 + 1$. Therefore in round $r+3$ every correct processor with *sentInits* = *false* must send *init* to all the processors (see lines 8–10). □

Definition For $a, b, c \in \mathbb{N}$, let $a \geq_c b$ denote the expression

$$(a < c \wedge a = b) \vee (a \geq c \wedge a \geq b \geq c - 1).$$

The next two lemmas provide a bound on the total number of votes that can be generated by all the faulty processors.

Lemma 3.3 *Let $\mathcal{E} = \mathcal{E}(\mathcal{C}, \mathcal{I}, \mathcal{M})$ be an execution in which no correct processor sends init messages in rounds $2, \ldots, r$, and let I be the number of correct processors whose input in $\mathcal{E}$ is 1. Suppose that at the end of round r the counter of some correct processor p is C. Then the total number of init messages received by all the correct processors from the faulty processors by round r is at least $(n-t-f)(C-I)$.*

Proof The proof is by induction on $(C - I)$. For $C \leq I$ the lemma obviously holds. For $C > I$, since in particular $C > 0$, it follows that p must have received (in round r or in some previous round) proposed values that are greater or equal to C from $n - t$ distinct processors, of which at least $n - 2t$ are correct. Let $r_0 < r$ be the smallest round by the end of which there exists a correct processor p_0 with *proposedCounter* $\geq C$. From the definition of r_0 it follows that in this round p_0 received at least $n - t$ possible values that are greater or equal to C, since otherwise it must have received at least $n - 2t$ proposed values greater or equal to C, of which at least $n - 3t$ must have originated from correct processors. Among these possible values, at least $n - t - f$ originated from correct processors that have *possibleCounter* $\geq C$. Let $p_1, \ldots, p_{n-t-f}$ denote these processors and let $r_i < r_0$, for $1 \leq i \leq n - t - f$, be the round at which the *possibleCounter* of p_i becomes greater or equal to C. It follows that p_i must have received an *init* message in round r_i. If $r_i > 1$ this *init* message must have come from some faulty processor, since according to the assumptions on $\mathcal{E}$ no correct processor sends such messages after round 1. Otherwise, $r_i = 1$, thus p_i must have received in the first round an *init* message from at least one faulty processor, since $C > I$. Let $m_i = (q_i, p_i, r_i, \mathit{init})$ denote an *init* message received by p_i in round r_i from some faulty processor q_i. Due to the previous argument such a message exists.

Consider a new execution $\mathcal{E}' = \mathcal{E}(\mathcal{C}, \mathcal{I}, \mathcal{M} \setminus \{m_1, \ldots, m_{n-t-f}\})$. Next, we prove a claim about the relationship between $\mathcal{E}$ and $\mathcal{E}'$, which will allow to complete the proof of the lemma by applying the induction assumption to $\mathcal{E}'$.

Claim *Let q be any correct processor. Let counter(s), proposedCounter(s), possibleCounter(s), PoV$[i](s)$ and PrV$[i](s)$, for $1 \leq i \leq n$, be the values of counter, proposedCounter, possibleCounter, PoV$[i]$ and PrV$[i]$ of q at the end of round*

s of $\mathcal{E}$, *respectively. Let* $counter'(s)$, $proposedCounter'(s)$, $possibleCounter'(s)$, $PoV'[i](s)$ *and* $PrV'[i](s)$ *be the values of counter*, *proposedCounter*, *possibleCounter*, $PoV[i]$ *and* $PrV[i]$ *of* q *in the end of round* s *of* $\mathcal{E}'$. *Then*

$$counter(s) \geq_C counter'(s), \tag{1}$$

$$proposedCounter(s) \geq_C proposedCounter'(s), \tag{2}$$

$$counter(s) < C \Rightarrow possibleCounter(s) \geq_C possibleCounter'(s). \tag{3}$$

Proof The proof is by induction on the round number s. For $s = 1$, (1) and (2) hold since at the end of the first round the *proposedCounter* and *counter* of every correct processor remain 0. If $possibleCounter(1) < C$ or if q is not one of the processors $p_1, \ldots, p_{n-t-f}$, then q receives the same number of *init* messages in $\mathcal{E}$ and $\mathcal{E}'$, thus $possibleCounter(1) = possibleCounter'(1)$. If $possibleCounter(1) \geq C$ and q is one of the processors $p_1, \ldots, p_{n-t-f}$, then in $\mathcal{E}'$ q receives one less *init* message than in $\mathcal{E}$, nevertheless $possibleCounter(1) \geq_C possibleCounter'(1)$.

Assuming that the claim holds for round s, we prove it for round $s + 1$.

If there exists a correct processor q' whose counter at the end of round s of $\mathcal{E}$ is at least C, by the induction the counter of q' in $\mathcal{E}'$ is at least $C - 1$. By Lemma 3.1, the counter of q at the end of round $s + 1$ is at least $C - 1$ in $\mathcal{E}'$, and at least C in $\mathcal{E}$. Since the *proposedCounter* of q can not be lower than its counter, we have shown that (1), (2) and (3) hold in round $s + 1$ as well.

Otherwise, the counter of every correct processor at the end of round s of $\mathcal{E}$ is less than C. Therefore (3) implies that before the array PoV is sorted, $PoV[i](s + 1) \geq_C PoV'[i](s + 1)$ for every link i that is connected to a correct processor. If link i of q is connected to a faulty processor, then $PoV[i](s + 1) = PoV'[i](s + 1)$, because the values sent by the faulty processors are the same in $\mathcal{E}$ and $\mathcal{E}'$. Similarly, (2) implies that before PrV is sorted, $PrV[i](s + 1) \geq_C PrV'[i](s + 1)$, for every $1 \leq i \leq n$. It is easy to see that these inequalities are unaffected by sorting the arrays. Therefore lines 14 and 15 of the algorithm (in round $s + 1$) leave the inequalities (1) and (2) correct.

We now prove that (3) holds in round $s + 1$. If $counter(s + 1) < C$, then (as just shown) the counter of q in $\mathcal{E}$ and $\mathcal{E}'$ is the same, thus line 20 is either executed in both $\mathcal{E}$ and $\mathcal{E}'$, or in none of them. In the later case we are done. Otherwise, there are three different possibilities. First, if $possibleCounter(s + 1) < C$, then the number of *init*s q receives is the same in $\mathcal{E}$ and $\mathcal{E}'$, so that $possibleCounter(s + 1) = possibleCounter'(s + 1)$. The second possibility is that $possibleCounter(s) \geq C$, in which case the claim follows from the induction assumption. The third possibility is when *possibleCounter* of q becomes greater or equal to C in round $s + 1$. In this case the number of *init*s q receives in $\mathcal{E}'$ is by at most 1 less than in $\mathcal{E}$, thus $possibleCounter'(s + 1) \geq C - 1$. This completes the proof of the claim. □

From the claim it is clear that the following two properties hold in $\mathcal{E}'$: (i) no correct processor sends *init* messages in rounds $2, \ldots, r$; and (ii) by the end of round r the counter of p in $\mathcal{E}'$ is at least $C - 1$. These two properties allow to apply the induction assumption on $\mathcal{E}'$. Since $\mathcal{E}'$ is obtained from $\mathcal{E}$ by removing $n - t - f$ *init* messages sent by the faulty processors to the correct processors, the lemma follows. □

Lemma 3.4 *Let I be the number of correct processors whose initial input in an execution $\mathcal{E}$ is 1. If no correct processor sends init messages in rounds $2,\ldots,r$, then by the end of round r of $\mathcal{E}$ the counter of every correct processor is at most $\lfloor (n-f)f/(n-t-f)\rfloor + I$.*

Proof Let C be the value of a counter of some correct processor by the end of round r. By Lemma 3.3, the total number of *init* messages sent by the faulty processors to the correct processors by round r is at least $(n-t-f)(C-I)$. Observe that the total number of links between the correct processors and the faulty processors is $(n-f)f$. Therefore the total number of *init* messages received by the correct processors from the faulty processors never exceeds $(n-f)f$. It follows that $(n-f)f \geq (n-t-f)(C-I)$, thus $C - I \leq \lfloor (n-f)f/(n-t-f)\rfloor$. □

Corollary 3.5 *By the end of round $3T+4$ either all the correct processors send init messages or at most t correct processors send init messages.*

Proof First, observe that if there are $t+1$ correct processors whose input value is 1 then by the end of round 3 the value of the counter of every correct processor is at least $t+1$. It follows that by round 4 all the correct processors send *init* messages. Otherwise, there are at most t correct processors with input value 1 (which send *init* messages in the first round of the algorithm). If there exists a correct processor that sends *init* messages in round r, where $3T+1 \geq r \geq 2$, then according to Lemma 3.2, by round $r+3$ all the correct processors send *init* messages. If such a processor does not exist, then no correct processor can send *init* messages after round $3T+1$ (which requires a counter that is at least $t+T+1$), because by Lemma 3.4 the counter of every correct processor is not greater than $\lfloor (n-f)f/(n-t-f)\rfloor + t \leq T+t$. □

Theorem 1 *The algorithm in Fig. 2 solves the binary simultaneous ABA problem for $n > 3t$.*

Proof (Agreement) Corollary 3.5 implies that by the end of round $3T+4$, either all or at most t correct processors send *init* messages. In the first case all the correct processors decide 1. Otherwise, each correct processor received *init* messages from at most $t+f$ processors. Since $n > 3t$, the decision rule of the algorithm (lines 21–24) implies that every correct processor decides 0.

(Validity) Consider the two possible cases. If the input of all the correct processors is 1, then all the correct processors send *init* messages in the first round, thus all the correct processors decide 1.

In the second case the input of all the correct processors is 0. The only *init* messages received in the first round by a correct processor originate from the faulty processors. Since there are at most t faulty processors, the *possibleCounter* by the end of the round is at most t. Therefore, in the second round after executing lines 14 and 15 of the algorithm, the *proposedCounter* and *counter* of every correct processor are at most t. Thus the IF expression in line 19 evaluates to FALSE, so that *possibleCounter* remains at most t. This last argument remains correct in all the subsequent rounds as well. Therefore, by the end of round $3T+4$ no correct processor sends *init* messages. In particular it follows that all the correct processors decide 0. □

An obvious optimization of the algorithm in Fig. 2 would be to refrain from sending the values of *possibleCounter* and *proposedCounter* if they were not updated. In this case, when a correct processor receives no value from another processor it assumes that it remained unchanged since the previous round.

3.3 Communication Complexity

To conclude this section, we analyze the communication complexity of the described algorithm. Let $C(r)$ denote the value of the highest *possibleCounter* or *proposedCounter* variables among all the correct processors in the end of round r. Observe that after executing line 14 of the algorithm in round $r+1$ the *proposedCounter* of any correct processor remains at most $C(r)$. Therefore, $C(r+1) - C(r) \leq n$, since any increase can be caused only by *init* messages received by some correct processor (in lines 17–20).

It follows that the values of *possibleCounter*s and of *proposedCounter*s of the correct processors in an execution of the algorithm can not grow beyond $n^{O(1)}$, since the number of rounds is $O(t)$. Therefore any such variable is represented by $O(\log n)$ bits. Since in each round every correct processor sends to all the processors its *possibleCounter* and *proposedCounter*, and possibly an *init* message ($O(1)$ bits), the total communication complexity of a single round is $O(n^2)$ messages and $O(n^2 \log n)$ message bits. Thus the communication complexity of the algorithm is $O(n^2 t)$ messages and $O(n^2 t \log n)$ message bits.

4 Achieving Early Stopping

In [14] it was shown that (in the standard model) the eventual BA problem can be solved in time proportional to the actual number of faulty processors f, rather than t. An eventual BA algorithm which has this property was called *early-stopping*.

In this section we present an early-stopping ABA algorithm which is a variation of the algorithm in Fig. 2. The only difference between the two algorithms is in the decision rule. While in the ABA algorithm in Fig. 2 the decision rule is applied only once (at the end of the last round), in the early-stopping algorithm, the decision rule (see Fig. 3) is applied at the end of every round to check if the processor is able to decide or stop. The new decision rule uses two new variables that need to be initialized as shown in lines 1, 2 of Fig. 3. Note that a correct processor that decides in round r does not necessarily stop in that round—it might need to run for a few more rounds to make sure all the other correct processors decide on the same value.

Lemma 4.1 *Let $C(r)$ be the value of the highest proposedCounter among all the active correct processors by the end of round r. Then the upperCounterBound of every active correct processor at the end of round r is at least $C(r)$.*

Proof The proof is by induction on the round number r**.** For $r = 1$ the lemma holds since $C(1) = 0$. Assume that the lemma is true for round r. For $r+1$, if $C(r) = C(r+1)$ then obviously the lemma is true in that round as well. If $C(r) < C(r+1)$,

```
INITIAL SETUP:
 1 stopRound := 3T + 4;
 2 upperCounterBound := 0;
DECISION RULE TO BE APPLIED AT THE END OF EVERY ROUND r:
 3 upperCounterBound := max(upperCounterBound, PoV[n − 2t]);
 4 IF init messages were received from n − t distinct processors THEN
 5    DECIDE 1;
 6    stopRound := min(stopRound, r + 3);
 7 ELSE IF r = 3T + 4 OR (upperCounterBound < t + r/3 − 1 AND r > 1) THEN
 8    DECIDE 0;
 9    STOP
10 IF r = stopRound THEN
11    STOP
```

Fig. 3 The decision rule for early-stopping Anonymous BA

let p be a correct processor whose *proposedCounter* is equal to $C(r+1)$ at the end of round $r+1$. Observe that in round $r+1$ the entry $PrV[n-2t]$ (in the sorted array PrV of p) can not be higher than $C(r)$. It follows that $PoV[n-t] = C(r+1)$ (in the sorted array PoV of p). This implies that by the end of the round the *upperCounterBound* of every correct processor is assigned a value which is at least $C(r+1)$ (see line 3 in Fig. 3). □

Consider the case in which there are $t+1$ correct processors with $input = 1$. Then in round 2 the value of *upperCounterBound* becomes at least $t+1$, so that no correct processor stops before round 4. In this case, as in the original ABA algorithm, all the correct processors send *init* messages by the end of round 4, and thus they all decide 1 by the end of that round.

Next we consider the case in which there are at most t correct processors with $input = 1$. Let r be the earliest round in which some correct processors stop and let p be one of the correct processors that stop in round r. If $r = 3T+4$, then we get an execution of the original algorithm, which was already shown to be correct in Theorem 1. Consider now the $r < 3T+4$ case. In rounds $1, \ldots, r$, all the correct processors are active, thus the lemmas proved in Sect. 4 are correct for these rounds of the early-stopping algorithm as well. This fact is used in the next two lemmas to prove that the early-stopping algorithm satisfies the agreement property of BA.

Lemma 4.2 *If p decides 1 then all the correct processors decide 1.*

Proof From the decision rule in Fig. 3 it follows that p decided 1 in round $r-3$. Therefore, by round $r-3$ p received *init* messages from $n-t$ distinct processors. At least one of these processors must be a correct processor with $input = 0$. Lemma 3.2 implies that by round r all the correct processors send *init* messages, therefore by the end of round r all the correct processors that did not decide so far must decide 1. Also, all the correct processors that decided before round r must have decided 1, otherwise there would have been a correct processor that stopped before round r, which contradicts the assumptions. Note that the above argument also shows that all the correct processors stop by round $r+3$. □

Lemma 4.3 *If p decides 0 then all the correct processors decide 0.*

Proof In the end of round r, the *upperCounterBound* of p must be less than $t + r/3 - 1$. Therefore, in the end of that round there must exist $t + 1$ correct processors whose *possibleCounter* is less than $t + r/3 - 1$. In addition, Lemma 4.1 implies that the *proposedCounter* of every correct processor in the end of round r is also less than $t + r/3 - 1$.

Since the *proposedCounter* of a correct processor can not be lower than its *counter*, it follows that by the end of round r the counter of any correct processor must be less than $t + r/3 - 1$. Therefore, in rounds $r - 2, r - 1, r$ no correct processor could have sent *init* messages or updated its *possibleCounter*. Also, no correct processor sent *init* messages in rounds $2, \ldots, r - 3$, because otherwise according to Lemma 3.2 by round r all the correct processors would have sent *init* messages, which would have caused p to decide 1.

Next we prove that the counters of all the correct processors remain lower than $t + r/3 - 1$ in the subsequent rounds as well, so that no correct processor will send *init* messages in round $r' \geq r$. This fact will complete the proof, since it shows that any correct processor will never get *init* messages from more than $t + f$ distinct processors (and thus will never decide 1). The proof is by induction on the round number r'. For $r' = r$ we showed that the following two properties hold in the end of the round: (i) there exist $t + 1$ correct processors which are either inactive or have *possibleCounter* that is less than $t + r/3 - 1$; (ii) the *proposedCounter*s and counters of all the correct processors are less than $t + r/3 - 1$. Now, for round $r' + 1$, properties (i) and (ii) imply that the *proposedCounter* as well as the counter of any correct processor must remain less than $t + r/3 - 1$ (see lines 14 and 15 in Fig. 2). This implies that no correct processor updates its *possibleCounter* in round $r' + 1$. □

Theorem 4.4 *The algorithm in Fig. 3 solves the binary ABA problem for $n > 3t$.*

Proof (Agreement) Lemmas 4.2 and 4.3 imply the agreement property.

(Validity) The case in which there exist $t + 1$ correct processors with *input* $= 1$ was already considered above. The proof for the case in which the *input* of all the processors is 0 is similar to that case in the proof of Theorem 1. □

Next, we analyze the number of rounds required by the algorithm. As explained above, if there are $t + 1$ correct processors with *input* $= 1$ then all the correct processors decide 1 by round 4, and thus must stop by round 7. Otherwise, there are at most t correct processors with *input* $= 1$, so that in an execution in which no correct processor sends *init* messages in rounds $2, \ldots, r$, the counter of every correct processor is not higher than $\lfloor (n - f) f/(n - t - f) \rfloor + t$ (see Lemma 3.4). In this case, from round 4 onwards the *possibleCounter* of a correct processor can be higher than its counter only as a result of *init* messages received from faulty processors. Thus, in this case the *possibleCounter* of any correct processor is not higher than $\lfloor (n - f) f/(n - t - f) \rfloor + t + f$.

If no correct processor with *input* $= 0$ sends *init* messages until round $3\lfloor (n - f) f/(n - t - f) \rfloor + 3f + 3$, then by the end of round $3\lfloor (n - f) f/(n - t - f) \rfloor +$

$3f+4$ the *upperCounterBound* of any correct processor is at most $\lfloor (n-f)f/(n-t-f)\rfloor + t + f$, which causes it to decide 0 and stop (see line 7 in Fig. 3).

Otherwise, there exists a correct processor q with $input = 0$ that sends *init* messages in round $2 \le r \le 3\lfloor (n-f)f/(n-t-f)\rfloor + 3f + 3$. Let p denote one of the processors that stop in the earliest round. It follows that p does not decide 0, because this contradicts the fact that q sent *init* messages in round r (as shown in Lemma 4.3). If p stops in round $r+2$ (or earlier), then all the correct processors stop by round $r+5$ (as shown in Lemma 4.2). If this is not the case, all the correct processors are active in rounds $r, \ldots, r+3$, so that according to Lemma 3.2, by round $r+3$ all the correct processors send *init* messages. It follows that all the correct processors decide 1 by round $r+3$ and stop by round $r+6$. Therefore, we have shown that every correct processor must stop after $\min(3\lfloor (n-t)t/(n-2t)\rfloor + 4,\ 3\lfloor (n-f)f/(n-t-f)\rfloor + 3f + 9)$ rounds.

The communication complexity analysis of the early-stopping algorithm is similar to that of the algorithm in Sect. 3. It shows that the algorithm uses $O(n^2 f)$ messages and $O(n^2 f \log n)$ message bits.

5 Conclusions

This paper presented a polynomial algorithm for the BA problem in a system with n anonymous processors of which up to t are faulty, where $n > 3t$. The presented algorithm runs in $3\lfloor (n-t)t/(n-2t)\rfloor + 4$ rounds. An early-stopping version that requires $\min(3\lfloor (n-t)t/(n-2t)\rfloor + 4,\ 3\lfloor (n-f)f/(n-t-f)\rfloor + 3f + 9)$ rounds was also presented. The communication complexity of the presented algorithms was shown to be $O(n^2 t \log n)$ bits. The methods used in these algorithms allow to develop a faster ABA algorithm for the $n > 5t$ case, which requires only $2\lfloor (n-t)t/(n-3t)\rfloor + 2$ rounds [22].

Our study of distributed anonymous systems with Byzantine failures can be extended in several directions. First, it is interesting to find the lower bound for the number of rounds required to achieve agreement in the anonymous model. We do know that this lower bound can *not* match the bound of the standard model [21]. A more difficult problem is to find an efficient algorithm that runs in the optimal number of rounds or close enough to that. An additional direction for further research is to solve the ABA problem in a more general class of network topologies.

Acknowledgements We wish to thank the referees for their comments. The research presented in this paper was supported in part by Israeli Council for Higher Education and by a grant from Dr. and Mrs. Silverstone, Cambridge, UK.

References

1. Angluin, D.: Local and global properties in networks of processors. In: Proc. 12th ACM Symposium on Theory of Computing (STOC), pp. 82–93 (1980)
2. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edn. Wiley-Interscience, New York (2004)

3. Attiya, H., Gorbach, A., Moran, S.: Computing in totally anonymous asynchronous shared memory systems. Inf. Comput. **173**(2), 162–183 (2002)
4. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure computation without authentication. In: Proc. 25th Annual International Cryptology Conference (CRYPTO), pp. 361–377 (2005)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proc. 20th ACM Symposium on Theory of Computing (STOC), pp. 1–10 (1988)
6. Boldi, P., Vigna, S.: Computing anonymously with arbitrary knowledge. In: Proc. 18th ACM Symposium on Principles of Distributed Computing (PODC), pp. 181–188 (1999)
7. Boldi, P., Vigna, S.: An Effective characterization of computability in anonymous networks. In: Distributed Computing, 15th International Conference (DISC), pp. 33–47 (2001)
8. Chaum, D.: Blind signatures for untraceable payments. In: Proc. CRYPTO, pp. 199–203 (1982)
9. Chaum, D.: Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA. In: Proc. EUROCRYPT, pp. 177–182 (1988)
10. Chaum, D., Crepeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proc. 20th ACM Symposium on Theory of Computing (STOC), pp. 11–19 (1988)
11. Cohen, J.D., Fischer, M.J.: A robust and verifiable cryptographically secure election scheme. In: Proc. 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 372–382 (1985)
12. Dolev, D., Strong, H.R.: Authenticated algorithms for Byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
13. Dolev, D., Fischer, M.J., Fowler, R., Lynch, N.A., Strong, H.R.: An efficient algorithm for Byzantine agreement without authentication. Inf. Control **52**(3), 257–274 (1982)
14. Dolev, D., Reischuk, R., Strong, H.R.: Early stopping in Byzantine agreement. J. ACM **37**(4), 720–741 (1990)
15. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Inf. Process. Lett. **14**(4), 183–186 (1982)
16. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. Distrib. Comput. **1**(1), 26–39 (1986)
17. Garay, J.A., Moses, Y.: Fully polynomial Byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. SIAM J. Comput. **27**(1), 247–290 (1998)
18. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Proc. 19th ACM Symposium on Theory of Computing (STOC), pp. 218–229 (1987)
19. Lamport, L., Schostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
20. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
21. Okun, M.: Agreement among unacquainted Byzantine generals. In: Distributed Computing, 19th International Conference (DISC), pp. 499–500 (2005)
22. Okun, M., Barak, A.: On anonymous Byzantine agreement. Leibniz Center TR 2004-2, School of Computer Science, The Hebrew University (2004)
23. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
24. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distrib. Comput. **2**(2), 80–94 (1987)
25. Srikanth, T.K., Toueg, S.: Optimal clock synchronization. J. ACM **34**(3), 626–645 (1987)
26. Toueg, S., Perry, K.J., Srikanth, T.K.: Fast distributed agreement. SIAM J. Comput. **16**(3), 445–457 (1987)
27. Turpin, R., Coan, B.A.: Extending binary Byzantine agreement to multivalued Byzantine agreement. Inf. Process. Lett. **18**(2), 73–76 (1984)
28. Yamashita, M., Kameda, T.: Computing on anonymous networks: Part I—characterizing the solvable cases. IEEE Trans. Parallel Distrib. Syst. **7**(1), 69–89 (1996)