Optimized Authenticated Self-synchronizing Byzantine Agreement Protocols

André Postma, Thijs Krol, Egbert Molenkamp
University of Twente, Department of Computer Science
P.O.Box 217, NL 7500 AE Enschede, the Netherlands
e-mail: postma@cs.utwente.nl, krol@cs.utwente.nl, molenkam@cs.utwente.nl

Abstract

In order to make a dependable distributed computer system resilient to arbitrary failures of its processors, deterministic Byzantine agreement protocols (BAPs) can be applied. Many BAPs found in literature require that communication takes place in synchronized rounds of information exchange and require that all correct processors know the start of the BAP and start the protocol simultaneously. It is hard to satisfy either or both requirements in a distributed system. As a consequence, it is hard to implement the above BAPs in a distributed system.

Authenticated self-synchronizing BAPs evade this problem by guaranteeing Byzantine Agreement while allowing arbitrary clock skew between the clocks of the processors and not requiring correct processors to know the start of the BAP. However, authenticated self-synchronizing BAPs require much communication overhead. Therefore, in this paper, we introduce so-called optimized authenticated self-synchronizing BAPs, that require fewer messages than the existing authenticated self-synchronizing BAPs.

1. Introduction

A dependable distributed computer system can withstand the failure of one or more of its processors, if every system service is provided by a set of replicated tasks running on different processors. However, malfunctioning client machines communicating with such a fault-tolerant system service may produce broadcast errors (i.e. send conflicting information to the different replicas of the system service) as a result of which inconsistency between the replicas of the system service may occur. This may lead to a system breakdown, even if the system does not contain more faulty processors than it is designed to tolerate [28].

In order to avoid inconsistency between the correct replicas of the system service, it must be ensured that the correct replicas agree on any information sent to them by clients. This can be achieved by having the replicas execute a so-called ***interactive consistency algorithm*** (***ICA***). An ICA runs on a set, *N*, of *N* processors p_i ($1 \le i \le N$), each of which holds an initial value v_i, and describes how the correct processors decide on a common value based on the initial values of all processors in *N*. In an ICA, every correct processor p_i distributes its initial value v_i to all other processors by means of a so-called ***Byzantine Agreement Protocol*** (***BAP***). In each BAP, the processor that distributes its initial value to the other processors is called the ***source***, the other processors are called the ***destination processors***. An extensive overview of BAPs is given in [1].

A BAP consists of a multicast process and a decision-making process. In the ***multicast process***, in a number of communication phases, a message value is transmitted from the source to all destination processors. In every communication phase, processors relay the messages they have received in the previous communication phase via communication links to other processors. In the ***decision-making process***, every destination processor calculates a decision about what the source has sent on basis of the message values of the messages it has received during the multicast process. Up to *T* processors in the system may behave maliciously (Notice that, since the source is one of the processors, it is also possible that the source behaves maliciously). Regardless which processors are faulty and which data was sent by the source, the protocol guarantees ***Byzantine agreement***, i.e. satisfaction of the ***interactive consistency conditions*** [2,3]:

IC1. *All correct processors agree on the data they think they have received from the source.*

IC2. *If the source is correct, the above-mentioned agreement equals the data sent by the source.*

What makes the problem of guaranteeing Byzantine Agreement in the presence of arbitrarily faulty processors so difficult, is the faulty processors' ability to behave maliciously. A faulty processor need not behave in the manner specified: it may refuse to send or relay messages, it may send or relay messages at arbitrary times (different from those specified in the BAP) and furthermore, it may corrupt the contents of messages.

1.1 Classification of BAPs

The design of a BAP depends on the assumptions made

0-8186-8212-4/97 $10.00 © 1997 IEEE

for the system. An important system model parameter is the synchrony of the system. In literature, synchronous and asynchronous systems are distinguished. In [4], several synchrony parameters are identified, resulting in different definitions of a synchronous system.

In this paper, we define a ***synchronous system*** as a system with the following two properties:

1. The rate at which the processor clock of any correct processor drifts from real-time, is bounded by a factor $(1+\rho)$. Such clocks are called ρ-***bounded clocks*** [5]. Processor clocks of faulty processors may run at arbitrary rates.
2. Furthermore, there exists a real-time upper bound, τ_{max}, on the time needed to communicate a message from a correct processor to another processor in the system.

Conversely, any system that is not synchronous is an ***asynchronous system.***

We assume that there also exists a real-time lower bound, τ_{min}, (with $0 \leq \tau_{min} \leq \tau_{max}$) on the time needed to communicate a message from a correct processor to another processor in the system. We further assume that the bounds τ_{min}, τ_{max}, and $(1+\rho)$ are a priori known by each correct processor in the system.

The definitions of synchronous and asynchronous systems given above are commonly used in research on Byzantine Agreement [4].

Different types of BAPs exist for both types of systems. For asynchronous systems, so-called ***randomized BAPs*** are designed. These randomized BAPs were first studied in [6,7,8], for an overview, see [1]. ***Deterministic BAPs*** (first defined in [2,3], see [1] for an overview) are designed for synchronous systems.

Whereas deterministic BAPs guarantee Byzantine Agreement within a finite number of communication phases, randomized BAPs may need an infinite number of communication phases to complete. Since, in a dependable distributed system, it is important to be able to guarantee Byzantine Agreement within finite time, our focus will be on *deterministic* BAPs in a *synchronous* system.

Besides the synchrony of the system, BAPs are characterized by other important parameters, such as the number of processors, N, the maximum number of faulty processors, T, for up to which the BAP guarantees Byzantine agreement, and the number of communication phases, K.

Furthermore, a distinction is made between authenticated and non-authenticated BAPs. In a so-called ***authenticated BAP***, the faulty processors' ability to behave maliciously is restricted by having every correct processor sign each message it relays with an unforgeable ***signature***. In a ***non-authenticated BAP***, no signatures are applied. Deterministic *non-authenticated* BAPs only exist for $N \geq 3T+1$ and $K \geq T+1$. These bounds are proved in [2,3] and [10] respectively. Deterministic *authenticated* BAPs can tolerate an arbitrary number of faulty processors. Since the problem is vacuous for $N \leq T+1$ [3], it is usually assumed that $N > T+1$. For $N > T+1$, authenticated BAPs need only satisfy $K \geq T+1$ (proved in [10]).

In authenticated BAPs, it is assumed that a correct processor's signature can not be forged and that any alteration of the contents of its signed messages can be detected. No assumptions are made about a faulty processor's signature. In particular, ***collusion*** among faulty processors is permitted, i.e., a faulty processor's signature may be forged by other faulty processors.

In practice, it is impossible to implement unforgeable signatures. However, they can be approximated by means of applying cryptographic techniques, e.g., public-key cryptosystems [12,13]. In this paper, we will assume that the signatures of correct processors are unforgeable. This is not a very restrictive assumption, since, by increasing the length of the cryptographic key, and taking into account the precautions suggested in [14], the probability of a correct processor's signature being forged can be made arbitrarily small [3, p.400]. Our focus will be on authenticated BAPs, because of their lower communication overhead and lower number of processors required, if compared to non-authenticated BAPs.

1.2. Guaranteeing Byzantine Agreement under less strict synchronicity assumptions

Many deterministic authenticated BAPs require a ***lock-step synchronous system***, i.e., they are based on the assumption of guaranteed communication in a network of perfectly synchronized processors (e.g., in [2,3,14,15,16]). They assume that communication takes place in synchronized rounds of information exchange and that all correct processors know the start of the BAP and start the protocol simultaneously.

In general, processors are not automatically synchronized, nor do the correct processors a priori know the start of a BAP. It seems easy to solve these problems by having all correct processors reach ***exact agreement*** about some point in time.

In a distributed system, however, reaching exact agreement about a common point in time is a difficult problem. Since every processor has its own local clock, processors do not have a common notion of time. Due to uncertain message delivery times, and processor clocks running at differing rates, any ***fault-tolerant clock synchronization algorithm*** (see [18] for an overview) can only establish *appromixate* synchronization between processor clocks. This makes it hard to have all correct processors reach exact agreement about a common point in time.

Exact agreement about some point in time can be obtained by means of a ***Distributed Firing Squad (DFS)***

algorithm [19]. However, a DFS-algorithm can not be used to solve the problem, since, in order to function correctly in the presence of arbitrarily faulty processors, a DFS-algorithm requires that the clocks of all correct processors run at the same rate, and that messages communicated between certain processors take a fixed time to be delivered. In a distributed system, it is hard to establish the amount of synchronism required by the DFS-algorithm, because, in practice, no two processor clocks run at the same rate [3], and message delivery times are uncertain.

In [20], it is proposed to run a BAP in order to have all correct processors obtain (as a result) exact agreement about a common point in time. Clearly, this method is not applicable here, because correct processors must already have obtained exact agreement about a common point in time *before* a BAP can be executed.

Since, in a distributed system, it is so difficult to have all correct processors reach exact agreement about a common point in time, some authenticated BAPs (see, e.g., [21]) have been designed which guarantee Byzantine Agreement provided that, before the BAP is started, the clocks of all correct processors are ***approximately synchronized***. In other words, before the start of the BAP, the ***clock skew*** (i.e. the difference in clock values [20]) between the clocks of all correct processors must be within certain bounds.

As stated before, correct processors may become approximately synchronized by running a ***fault-tolerant clock synchronization algorithm***. A great number of fault-tolerant clock synchronization algorithms have appeared in the literature (see [18] for an overview). Roughly, these algorithms can be divided into two groups: probabilistic and deterministic algorithms, respectively. Although *probabilistic* fault-tolerant clock synchronization algorithms (e.g., [22]) reach better average synchronization between processor clocks, there is always a probability that the clocks cannot be synchronized within finite time [23], so these algorithms are not applicable here. Deterministic fault-tolerant clock synchronization algorithms can guarantee approximate synchronization within finite time. These algorithms require that $N \geq 3T+1$, unless authentication is used or there is a bound on the rate at which messages can be generated (proved in [24]). In general, it is impossible to indicate a priori an upper bound on the rate at which messages can be generated, so either we must require that $N \geq 3T+1$, or authentication must be used.

Clock synchronization algorithms that use authentication (e.g., the algorithm in [25]) can tolerate an arbitrary number of failures, provided that the clocks of all correct processors are initially approximately synchronized. To integrate (join) new or repaired processors such that their clocks become synchronized with those of all the other correct processors, a so-called ***bounded joining algorithm*** [25] is required. A necessary condition for a bounded joining algorithm to be guaranteed to succeed is that a majority of the processors in the system be correct (i.e. $N \geq 2T+1$) (proved in [25]).

So, for the clocks of all correct processors to be approximately synchronized in the presence of up to T faulty processors, which may differ for each run of the BAP in [21], a deterministic fault-tolerant clock synchronization algorithm must be periodically executed, and the system must contain at least a majority of correct processors.

To evade the difficulties described above, in [26], we have defined a new class of ***authenticated self-synchronizing BAPs***, that guarantee Byzantine agreement in a synchronous system *without* the requirement that all correct processors must be approximately synchronized before the start of the BAP (i.e. the clock skew between the processor clocks of the processors may be arbitrary), and without requiring the correct processors to know the start of the BAP. These BAPs can guarantee Byzantine Agreement for any number of faulty processors. The required amount of protocol synchronization between the correct processors in the system is established during execution of the BAP itself by means of ***diffusion induction*** [21]. Informally, upon receipt of the first valid message, any correct processor implicitly informs all processors that did not receive this message of the start of the BAP, by relaying the message to them. This guarantees that all processors that had not been informed about the start of the BAP, are implicitly informed of it upon receipt of the message. For more details, see [26].

Unfortunately, authenticated self-synchronizing BAPs are inefficient with regard to the required communication overhead. In this paper, we will therefore define a new class of ***optimized authenticated self-synchronizing BAPs*** (based on the BAPs in [26]), that require fewer messages to be communicated during the BAP.

Another assumption that is commonly made for deterministic authenticated BAPs (e.g., in [2,3,14,15,16,26]), is that every correct processor immediately notices when its processor clock reaches a certain predefined value (in particular, a clock value at which a communication phase ends). However, every processor clock is a discrete clock with finite precision, and moreover, a processor may not continuously check the value of its processor clock. Therefore, it is more realistic to assume that a processor can only determine within certain bounds that its processor clock is at or beyond a certain value. The BAPs presented in this paper will be based on this so-called ***imprecise clock value measurement assumption*** (Assumption A3 in Section 5).

1.3. Classification of authenticated BAPs with regard to synchronicity assumptions

A vast amount of literature has appeared on deterministic authenticated BAPs, based on varying synchronicity assumptions. In Table 1, we have classified several authenticated BAPs according to assumptions made with regard to the required tightness of synchronization between the processor clocks of correct processors in the system (i.e. the maximally allowed clock skew), and the tightness of synchronization required at the start of the protocol (protocols that require a synchronous start in all correct processors vs. self-synchronizing protocols). Notice that, of all the BAPs referred to in Table 1, the authenticated BAPs in [26] and the ones in this paper, which are self-synchronizing and which allow arbitrary clock skew between the clocks of the processors in the system are **the least restrictive** deterministic BAPs with regard to the synchronicity that is required.

Table 1: Classification of authenticated BAPs with regard to synchronicity assumptions

	start of protocol must be synchronized	**protocol is self-synchronizing**
processor clocks must be exactly synchronized (*no clock skew allowed*)	[2], [3], [14], [15], [16] and many others	
processor clocks must be approximately synchronized (*bounded clock skew allowed*)		[21]
processor clocks need not be synchronized (*arbitrary clock skew allowed*)		[26], optimized authenticated self-synchronizing BAPs in this paper

1.4. Overview

The rest of this paper is structured as follows. In Section 2, we discuss the notion of path information. In Section 3, some definitions are given. In Section 4, we describe our optimized authenticated self-synchronizing BAPs. The assumptions we make for these BAPs are given in Section 5. In Section 6, for some practical values of T, N, ρ, τ_{min}, and τ_{max}, the optimized self-synchronizing BAPs are compared with the self-synchronizing BAPs in [26].

2. Path information

Lamport et al. were the first to formulate the problem of Byzantine Agreement [2] and to define some BAPs for lock-step synchronous systems to solve this problem [3]. Since then, a lot of more efficient solutions to this problem have appeared, many of them, however, being based on the BAPs formulated in [3].

The BAPs in [3] are based on the assumption that for every valid message m (a definition of a valid message will be given in Section 3) received by a correct processor c, c knows the ***path*** (i.e. the sequence of processors) along which m travelled from the source to c. This so-called ***path knowledge assumption*** is generally used in other BAPs, too (e.g., in [1,14,15,16,17]). It is needed in the multicast process, to enable a correct processor to determine for any received valid message m the set of processors that did not yet receive m, and to relay m to these processors.

In arbitrary synchronous systems (with arbitrary ρ,τ_{min} and τ_{max}), the path knowledge assumption does not automatically hold. In order to enable correct processors to obtain the required information, in the BAPs described in this paper, so-called ***path information*** is included in each valid message. If the path information is correct, it describes the path along which the message travelled from the source to the current processor. However, there is no easy way to check whether the path information of a message is correct or not, since malicious processors may undetectably corrupt the path information of messages they relay. We can only check if the path information satisfies several properties (listed in Section 3). Path information satisfying these properties is called ***valid path information***.

By having every correct processor sign the path information of each message with an ***unforgeable signature***, the effects of corruption of this ***authenticated path information*** by malicious processors can be restricted.

We assume that a correct processor's signature can not be forged, and any alteration of the contents of its signed messages can be detected. No assumptions are made about a faulty processor's signature. In particular, any faulty processor's signature may be forged by other faulty processors (i.e., ***collusion*** among faulty processors is allowed).

Provided that signatures of correct processors can not be undetectably forged by malicious processors, the authenticated path information of a valid message m *only* contains the signature of a correct processor c, if processor c *did actually receive* message m. Then, the set of processors, the signature of which is absent in the authenticated path information of m, includes all correct processors that did not yet receive message m. Thus, provided that signatures of correct processors can not be undetectably forged by malicious processors, it is possible to deduce from the authenticated path information of a valid message m a set of processors that includes all correct processors that did not yet receive m.

In this paper, we assume that in our BAPs every valid message m consists of two parts: the ***message value***, i.e. the information that the sender of m wanted to communicate, and authenticated path information of m. By integration of a message digest [27] of the message value of a message *m* in the authenticated path information of *m*, malicious processors are prevented from undetectably combining the message value of one message with the path information of a different message.

3. Definitions

A BAP runs on a system consisting of a set, *N*, of *N* processors interconnected by a network. One of the processors is the source. The source wants to communicate a value to the other processors (the destination processors). In general, the source starts the protocol by multicasting a message to some or all of the destination processors in the network. At receipt of the first valid message from the BAP, a correct destination processor will start its own sub-protocol. In order to prevent confusion, we will refer to the protocol as a whole as the ***BAP***, and to any 'sub-protocol' of the BAP running on a certain destination processor *p* as the ***sub-BAP of processor p***.

In the paper, we use the following definitions:

D1. A ***path-valid message*** is a message with valid path information, i.e. path information describing a path of length $\leq T+2$ in which every processor appears at most once, and the sender and receiver of the message are the last two processors in the path.

D2. A ***valid message*** is a correctly authenticated and path-valid message.

D3. An ***invalid message*** is a message that is not valid.

D4. A message *m* is ***received in time*** in a correct processor *c* if it arrives in *c* in or before communication phase *i* ($1 \leq i \leq T+1$) of *c*'s sub-BAP, and the path information of *m* describes a path containing at least $i+1$ processors.

D5. A message *m* is ***received too late*** in a correct processor *c* if it arrives in *c* after *c* has concluded communication phase *i* ($1 \leq i \leq T+1$) of its sub-BAP, and the path information of *m* describes a path containing at most $i+1$ processors.

D6. A message *m*, sent directly from the source to a destination processor *d* is ***sent in time***, if, after the source started the BAP (by sending the first message of the BAP), *m* arrives in *d* within a real-time interval of length between τ_{min} and τ_{max}. Similarly, a message *m*, relayed by a processor *j* in communication phase *i* of its sub-BAP to another processor *k* is ***sent in time*** if *m* arrives in *k* within a real-time interval of length between τ_{min} and τ_{max} after *j* started communication phase *i* of its sub-BAP.

4. A description of an optimized authenticated self-synchronizing BAP

In [26], authenticated self-synchronizing BAPs have been described. Clearly, these BAPs are inefficient with regard to the number of messages communicated, since in these BAPs, every valid message *m*, that is received by a correct processor is relayed to all destination processors, the signature of which is absent in the path information of *m* (i.e. to all correct destination processors that did not yet receive *m* as well as to all faulty processors, the signature of which is absent in the path information of *m*).

In the literature, several ***optimized BAPs*** have been designed, that are more efficient with regard to the number of messages exchanged (e.g., in [15,16]) if compared to the BAPs in [3]. In the optimized protocols, received messages are only relayed to a large enough subset of processors, instead of to all processors that did not yet receive the message (e.g., in [3]). However, the proposed optimized protocols in [15,16] assume a lock-step synchronous system and are based on the assumption that all correct processors start the protocol simultaneously.

In this section, we present an ***optimized authenticated self-synchronizing BAP*** (based on the BAPs in [26]), which works in any synchronous system, which does not require all processors to start the protocol simultaneously and which requires less communication overhead than the BAPs in [26]. However, the reduction in the number of messages communicated in these optimized authenticated self-synchronizing BAPs is obtained at the cost of increased lengths of the $T+1$ communication phases of the sub-BAPs (for $T \geq 2$), if compared to the BAPs described in [26]. See Section 6 for details.

In an optimized authenticated self-synchronizing BAP, a correct processor relays every valid message it receives to a large enough subset of processors. We assume that these subsets are selected according to some predefined rules, known by all correct processors. On basis of these so-called ***subset selection rules*** of a BAP, any correct processor expects that the path described in the path information of the first valid message it receives from the BAP, has a length greater than or equal to a certain *k* ($2 \leq k \leq T+1$). Faulty processors may relay received messages to arbitrary subsets of processors, different from those defined by the subset selection rules.

By relaying valid messages only to a subset of processors, protocol synchronization between all correct processors, as described in Section 1.2, may not be guaranteed, since the subset selection rules may be chosen such that some correct processors never receive a valid message. Therefore, we assume that the subset selection rules are chosen such that every valid message from the source is relayed to all correct processors within *T* relay steps.

In order to be able to guarantee this assumption, the subset selection should be selected such that the subsets fulfil the following two requirements:

SSR1. For any timely received valid message *m*, with path information describing a path containing i processors ($2 \le i \le T$), the subset of processors, which m is relayed to, should contain at least $T+1$ processors that are not in the path information of m. If the number of processors that are not in the path information is less than $T+1$, then the subset should contain all these processors.

SSR2. For any timely received valid message m, with path information describing a path containing $T+1$ processors, the subset of processors, which m is relayed to, should contain all processors that are not in the path information of m.

An optimized authenticated self-synchronizing BAP can be described as follows:

1. The source s signs and sends a message m to every processor i selected by the subset selection rules of s. The source accepts a copy of its own message in order to use it in the decision-making process.

For each processor $i \in N$:

2. If processor i timely receives a valid message m which i expects on basis of the subset selection rules of the BAP, and i has not received a message with path information describing the same path before, i accepts m. If message m is the first valid message that i receives, i starts the first communication phase of its sub-BAP at receipt of m. Processor i now calculates at which clock values of its processor clock each of the T+1 communication phases of its sub-BAP ends. Processor i is able to do this, since i can calculate a priori the length of all communication phases of its sub-BAP, viewed from i's processor clock. For all k, with $1 \le k \le T+1$, for any correct processor i, let L_k be the length of communication phase k, as viewed from i. Then:

$$L_1 = (\tau_{max} - \tau_{min}) \cdot (1+\rho)$$

$$L_2 = (\tau_{max} - \tau_{min}) \cdot (1+\rho)^3 + (\Delta + T \cdot \tau_{max} + \tau_{min}) \cdot (1+\rho).$$

$$\forall k \in [3, T+1] : L_k = [L_{k-1} \cdot (1+\rho) + \Delta] \cdot (1+\rho).$$

3. If i receives an invalid message, a valid message that is received too late or a valid message with path information describing a path that is identical to a path described in path information of a message that i received before, or a valid message which i does not expect on basis of the subset selection rules of the BAP, i will reject the message.

4. If i accepted m, and the path information of m describes a valid path with less than T+2 processors, for every processor j to which i should relay m according to i's subset selection rules, i signs message m and i relays m to j, immediately after m was received by i.

5. After the multicast process of its sub-BAP, i decides on basis of the message values of the messages it has accepted.

All correct processors are assumed to execute the above-described protocol. The behaviour of faulty processors, however, may deviate arbitrarily from the described protocol.

5. Assumptions for optimized authenticated self-synchronizing BAPs

The assumptions we make for our optimized authenticated self-synchronizing BAPs are:

A1. There exist fixed known upper and lower bounds (τ_{max} and τ_{min} respectively) on the time required to communicate a message from one correct processor to another processor in the system. Furthermore, $0 \le \tau_{min} \le \tau_{max}$.

A2. For any processor p and any real time t, let $C(p, t)$ be the value of p's clock at time t. Then, for any correct processor p in the system, and any two points t_1 and t_2 in real-time, we assume there exists a $\rho \ge 0$ (known by all correct processors), such that:

$$\frac{1}{1+\rho} \le \frac{C\text{[redacted]}}{t_1 - t_2} \le (1+\rho)$$

Thus, we assume that the processor clocks of correct processors are ρ-bounded. Notice that the value $C(p,t)$ of the clock of a correct processor p may deviate arbitrarily from real-time t (Thus, the clock skew between processor clocks of the processors in the system is unbounded).

A3. There exists a real-time bound Δ on clock value measurement uncertainty, known by all correct processors. The first time any correct processor p detects that its clock is beyond clock value v, it is assumed that p's clock indicated v at most Δ ago.

A4. A processor may concurrently execute different BAPs. We assume that messages of different BAPs can be distinguished, such that every correct processor can determine when it receives the first valid message of a new BAP.

A5. Processors communicate by means of a reliable point-to-point network with perfect communication links. This assumption is justified by the fact that we can model a link failure as a failure of one of its adjacent processors [18].

A6. Every valid message m contains authenticated path information. Knowledge of the path information of m is sufficient to determine a set of processors that includes all correct processors that did not yet receive m.

A7. In a BAP, correct processors relay the valid messages they accept to a large enough subset of processors according to the subset selection rules for that BAP.

A8. Every correct processor knows the subset selection rules of the BAP.

A9. The subset selection rules satisfy SSR1 and SSR2.

A10. If the source is correct, in every destination processor d selected by the subset selection rules of the source, a valid message from the source arrives in d within a real-time interval of length between τ_{min} and τ_{max} after the source started the BAP. ❑

No assumptions are made about the behaviour of faulty processors. They may send arbitrary messages at arbitrary times, or refuse to relay received messages. Their processor clocks may run at arbitrary rates. Furthermore, faulty processors may collude, i.e. they may use each other's signature to sign messages. However, we assume that faulty processors can not prevent correct processors from meeting the assumptions stated in this section. In particular, we assume guaranteed communication from a correct processor to other processors in the system (i.e. satisfaction of A1). In practice, there is always a possibility that faulty processors continuously generate new messages, as a result of which network congestion may occur. Solving this problem is nontrivial, but beyond the scope of this paper.

Analogously as it is done in [26], it can be proved that the BAPs presented in Section 4 guarantee Byzantine Agreement in a synchronous system of N processors, up to T of which may behave maliciously, provided that the above-stated assumptions hold. The proofs can be found in [29]. In this paper, the proofs have been omitted due to space limitations.

6. A comparison with existing self-synchronizing BAPs

In this section, for some practical values of T, N, ρ, τ_{min}, and τ_{max}, we compare the optimized self-synchronizing BAPs with the self-synchronizing BAPs in [26], which we will refer to as non-optimized self-synchronizing BAPs[1]. For the latter BAPs, the lengths of the various communication phases of the protocols are given by:

$$L_1 = (\tau_{max} - \tau_{min}) \cdot (1+\rho)$$

$$L_2 = [(\tau_{max} - \tau_{min}) \cdot (1+\rho)^3 + 2\tau_{max} \cdot (1+\rho)]$$

$$\forall\, i \in [3, T+1] : L_i = L_{i-1} \cdot (1+\rho)^2$$

We choose $\rho = 10^{-4}$, $\tau_{min} = 0$ ms and $\tau_{max} = 20$ ms. These values are realistic for an ATM-network. Since the imprecise clock value measurement assumption is not taken into account in the BAPs in [26] (i.e. $\Delta = 0$ in these BAPs), we choose $\Delta=0$ in the optimized BAPs, too.

1. We only compare the optimized self-synchronizing BAPs with the BAPs in [26], since other BAPs discussed in this paper are based on a system model with more restrictive synchronicity assumptions.

In Table 2 and 3, for some values of T and N, the required number of messages for both non-optimized and optimized BAPs are given. We assume that, in optimized BAPs, in every communication phase except the last one, a valid message is relayed to a subset of T+1 processors. In the last communication phase, the message is relayed to all processors that did not receive it.

The results show that for $N > T+2$, optimized BAPs require fewer messages than non-optimized ones. In Table 4, for some values of T, we have compared the protocol execution time of both types of BAPs. We see that, for $T \geq 2$, in an optimized BAP, the execution time of a sub-BAP is greater than or equal to the execution time of a sub-BAP in a non-optimized BAP.

Table 2: Required number of messages for T=1

N	#mess. in non-optimized BAPs	#mess. in optimized BAPs
3	4	4
4	9	6
5	16	8
16	256	30

Table 3: Required number of messages for T=2

N	#mess. in non-optimized BAPs	#mess. in optimized BAPs
4	15	15
5	40	30
6	85	39
16	2955	129

Table 4: Protocol execution time of sub-BAP (in ms)

T	Execution time in non-optimized BAPs	Execution time in optimized BAPs
1	80.014	40.010
2	140.038	140.038
3	200.074	260.092
4	260.122	420.770

7. Conclusion

Authenticated self-synchronizing Byzantine Agreement Protocols (BAPs) evade certain restrictions regarding the synchronicity of the system that are required by existing authenticated BAPs in the literature. The optimized authenticated self-synchronizing BAPs described in this paper require lower communication overhead than previously described authenticated self-synchronizing BAPs. Provided that the assumptions A1 through A10 (in Section 6) hold, execution of such an optimized authenticated self-synchronizing BAP in a fully-connected system of N processors, up to T of which may behave maliciously, guarantees Byzantine Agreement.

8. References

[1] Barborak, M., et al., The Consensus Problem in Fault-Tolerant Computing, in: **ACM Computing Surveys**, Vol.25, No.2, June 1993, pp. 171-220.

[2] Pease, M., Shostak, R., and Lamport, L., Reaching agreement in the presence of faults, in: **Journal of the ACM**, Vol.27, No.2, April 1980, pp.228-234.

[3] Lamport, L., et al., The Byzantine Generals Problem, in: **ACM Transactions on Programming Languages and Systems**, Vol.4, No.3, July 1982, pp. 382-401.

[4] Dolev, D., et al., On the Minimal Synchronism Needed for Distributed Consensus, in: **Journal of the ACM**, Vol.34, No.1, January 1987, pp.77-97.

[5] Lundelius Welch, J., and Lynch, N., A New Fault-Tolerant Algorithm for Clock Synchronization, in: **Information and Computation**, Vol.77, 1988, pp. 1-36.

[6] Rabin. M.O., Randomized Byzantine Generals, in: **Proceedings of the 24th Symposium on Foundations of Computer Science**, 1983, pp.403-409.

[7] Ben-Or, M., Another advantage of free choice: Completely asynchronous agreement protocols, in: **The 2nd Annual ACM Symposium on Principles of Distributed Computing**, ACM, 1983, pp.27-30.

[8] Bracha, G., and Toueg, S., Resilient consensus protocols, in: **The 2nd Annual ACM Symposium on Principles of Distributed Computing**, ACM, 1983, pp.12-26.

[9] Fischer, M., Lynch, N., and Paterson, M., Impossibility of distributed consensus with one faulty process, in: **Journal of the ACM**, Vol.32, No.2, 1985, pp.374-382.

[10] Fischer, M., and Lynch, N., A lower bound for the time to assure interactive consistency, in: **Information Processing Letters**, Vol.14, No.4, 1982, pp.183-186.

[11] Vanstone, S.A., and van Oorschot, P.C., **An Introduction to Error Correcting Codes with Applications**, Kluwer Academic Publishers, Boston, 1989.

[12] Diffie, W., and Hellman, M.E., New directions in cryptography, in: **IEEE Transactions on Information Theory**, IT22(6), November 1976, pp.644-650.

[13] Rivest, R., et al., A method for obtaining digital signatures and public-key cryptosystems, in: **Communications of the ACM**, Vol. 21, 1978, pp.120-126.

[14] Gong, L., et al., Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults, in: **Proceedings of IFIP-DCCA-5**, Urbana-Champaign, 1995, preprints, pp.79-90.

[15] Dolev, D., and Strong, H.R., Authenticated algorithms for Byzantine agreement, in: **Siam Journal on Computing**, Vol.12, No.4, 1983, pp. 656-666.

[16] Postma, A., and Krol, Th., Interactive Consistency Algorithms Based on Authentication and Error-Correcting Codes, in: **Proceedings of IFIP-DCCA-5**, Urbana-Champaign, 1995, preprints, pp.91-101. (*)

[17] Dwork, C., Lynch, N., and Stockmeyer, L., Consensus in the presence of partial synchrony, in: **Journal of the ACM**, Vol.35, No.2, April 1988, pp.288-323.

[18] Simons, B., Lundelius Welch, J., and Lynch, N., An Overview of Clock Synchronization, in: Simons, B., and Spector, A. (Eds.), **Fault-Tolerant Distributed Computing**, Springer Verlag, Berlin, 1990, pp.84-96.

[19] Coan, B.A., Dolev, D., Dwork, C., and Stockmeyer, L., The Distributed Firing Squad Problem, in: **The 17th ACM Symposium on the Theory of Computing**, ACM, New York, 1985, pp.335-345.

[20] Suri, N., Hugue, M.M., and Walter, C.J., Synchronization Issues in Real-Time Systems, in: **Proceedings of the IEEE**, Vol.82, No.1, January 1994, pp.41-54.

[21] Cristian, F., Aghili, H., Strong, R., and Dolev, D., Atomic broadcast: From simple message diffusion to Byzantine agreement, in: **Information and Computation**, Vol.118, 1995, pp.158-179. (Earlier version appeared in: **Proceedings of FTCS-15**, Ann Arbor, 1985, pp.200-206)

[22] Cristian, F., Probabilistic clock synchronization, in: **Distributed Computing**, Vol. 3, 1989, pp.146-158.

[23] Jalote, P., **Fault Tolerance in Distributed Systems**, Prentice Hall, New Jersey, 1994, pp. 97-99.

[24] Dolev, D., Halpern, J.Y., and Strong, H.R., On the possibility and Impossibility of Achieving Clock Synchronization, in: **Journal of Computer and System Sciences**, Vol.32, No.2, pp.230-250, 1986.

[25] Dolev, D., et al., Dynamic Fault-Tolerant Clock Synchronization, in: **Journal of the ACM**, Vol.42, No.1, 1995, pp.143-185. (An earlier version of this paper from the same authors was entitled: Fault-Tolerant Clock Synchronization, and appeared in: **Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing**, ACM, New York, 1984, pp.89-102)

[26] Postma, A., and Krol, Th., Interactive Consistency in Quasi-Asynchronous Systems, in: **Proceedings of ICECCS'96**, Montréal, Canada, 1996, pp.2-9. (*)

[27] Mullender, S.J., **Distributed systems**, 2nd Edition, Addison-Wesley, New York, 1993, p.533.

[28] Krol, Th., **A Generalization of Fault-Tolerance Based on Masking**, Ph.D.thesis, Eindhoven University of Technology, 1991.

[29] Postma, A., Krol, Th., and Molenkamp, E., **Optimized Authenticated Self-Synchronizing Byzantine Agreement Protocols**, CTIT Technical Report, University of Twente, 1997. (*)

(*) Available on: http://wwwspa.cs.utwente.nl/aid/fade/fade.html