Interactive Consistency in Quasi-Asynchronous Systems

André Postma, Thijs Krol
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

Abstract

In order to make a dependable distributed computer system resilient to arbitrary failures of its processors, deterministic interactive consistency algorithms (ICAs) are required. Thus far, in order to guarantee interactive consistency, all ICAs found in the literature require that all correct processors in the system start the algorithm simultaneously. In a distributed system, it is hard to satisfy this requirement. Therefore, in this paper, we describe a new class of self-synchronizing ICAs that guarantee interactive consistency without the above-mentioned requirement.

1. Introduction

A dependable distributed computer system can be made resilient to arbitrary failures of one or more of its processors, by ensuring that every system service is provided by a set of replicated tasks running on different processors. However, malfunctioning client machines that communicate 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.

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. In the presence of arbitrarily faulty (or ***malicious***) processors, this is achieved by having every correct replica communicate every input value received from a client with every other replica in the service by means of a so-called ***interactive consistency algorithm*** (or Byzantine Generals algorithm), which will be defined below. In such an interactive consistency algorithm (or ICA for short), the correct replica that wants to communicate an input value from a client represents the ***source*** that communicates a value to a set of ***destination processors***, being the other replicas in the service. An extensive overview of interactive consistency algorithms is given in [1].

An ICA runs on a system consisting of N processors, one of which is the source. An ICA consists of a broadcast process and a decision-making process. In the ***broadcast process***, in a number of communication phases, a message value is transmitted from the source to a set of destination processors. In the first communication phase, the source sends its message value to all other processors. In every subsequent communication phase, processors relay messages which they received in the previous communication phase to other processors via so-called communication links. In the ***decision-making process***, every destination processor calculates a decision about what the source has sent on basis of all messages it has received during the broadcast process. Up to T processors in the system may behave maliciously, i.e. send conflicting messages to different processors. Regardless which processors are faulty and which data was sent by the source, the algorithm guarantees ***interactive consistency***, i.e. satisfaction of the so-called ***interactive consistency conditions*** [2,3]:

- ❑ *The correct processors agree on the data they think they have received from the source.*
- ❑ *If the source is correct, the above-mentioned agreement equals the data sent by the source.*

Several prime parameters that characterize ICAs are the number of processors, N, the maximum number of faulty processors, T, for which the ICA still guarantees interactive consistency, and the number of communication phases K.

Interactive consistency can only be guaranteed by a *deterministic* ICA in a *synchronous* system. The minimal amount of synchronism needed in a system to guarantee interactive consistency is investigated in [4]. Synchronous deterministic non-authenticated ICAs only exist for $N \geq 3T+1$ and $K \geq T+1$, whereas synchronous deterministic authenticated ICAs have to satisfy $N \geq T+1$ and $K \geq T+1$.

Thus far, all deterministic ICAs found in the literature (e.g. in [1,3,5,6]) assume that *all correct processors start the algorithm simultaneously.* Without this assumption, interactive consistency can not be guaranteed with existing ICAs.

However, in general, the correct processors do not a pri-

0-8186-7614-0/96 $5.00 © 1996 IEEE

ori know the start of an ICA. This problem can be solved by having all correct processors agree on some point in time at which the ICA is started.

In a distributed system, agreeing on a common point in time is a difficult problem. Since every processor has its own local processor clock, processors do not have a common notion of time. In practice, as stated in [3], no two processor clocks run at exactly the same rate, as a result of which the clocks will drift away from each other. Several fault-tolerant clock synchronization algorithms exist (see [7] for an overview), by means of which the processor clocks can be synchronized within certain bounds. Unfortunately, in practice, the difference in synchronization between any two processor clocks will never be zero. This makes it hard to have all correct processors agree on a common point in time.

The problem can be solved by a Distributed Firing Squad (DFS) algorithm [8]. A DFS-algorithm runs on a system of N processors, up to T of which may behave maliciously, and has the following two properties:

- ❑ *If any correct processor receives a message to start a DFS-synchronization, then at some future time all correct processors will "fire" (i.e. enter a special state).*
- ❑ *The correct processors all fire simultaneously.*

However, in order to function correctly in the presence of up to T arbitrarily faulty processors, a DFS-algorithm requires that the clocks of all correct processors run at the same rate, and that message delivery times between either two correct or two faulty processors are fixed.

In practice, as stated in [3], no two processor clocks run at exactly the same rate. Furthermore, in a distributed system, there will always be some uncertainty in message delivery times. The degree of synchronism required in a DFS-algorithm can therefore only be realized in tightly-coupled systems with a centralized clock.

A distributed system can better be modelled as a ***quasi-asynchronous system***, i.e. a system with the following properties:

1. The amount by 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*** [9]. Processor clocks of faulty processors may run at arbitrary rates.
2. Furthermore, there exist a real-time upper and lower bound (τ_{max} resp. τ_{min}) on the time needed to communicate a message from a correct processor to another processor in the system.

In this paper, we define a new class of self-synchronizing ICAs that guarantee interactive consistency in a quasi-asynchronous system *without* the assumption that all correct processors start the algorithm simultaneously. The required degree of algorithm synchronization between correct processors is acquired by execution of the algorithm itself. We define the so-called ICA requirements, i.e. the requirements that are sufficient for an ICA to guarantee interactive consistency, and we prove that our algorithms satisfy the ICA requirements. Finally, we describe how existing authenticated deterministic ICAs found in the literature (e.g. in [1,3,5,6]) can be adapted to satisfy the ICA requirements in a quasi-asynchronous system.

2. An existing authenticated ICA

Lamport et al. were the first to formulate the interactive consistency problem [2] and some ICAs for fully-connected synchronous systems to solve the problem [3]. Since then, a lot of more efficient (though more difficult) solutions to the problem have appeared, most of them, however, being based on the ICAs formulated in [3].

Fulfilling the interactive consistency conditions seems deceptively simple. What makes the ICAs so difficult, is the faulty processors' ability to behave maliciously. The problem becomes easier to solve if this ability can be restricted. One way to do this is to allow the processors to sign each message they relay with an ***unforgeable signature***. An ICA that uses this method is called an ***authenticated ICA***. 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, we allow a faulty processor's signature to be forged by another faulty processor, thereby permitting ***collusion*** among the faulty processors [3].

The algorithm *SM(T)* (SM for 'Signed Messages') in [3] and presented below is an example of an authenticated ICA. Our self-synchronizing ICAs will be based on the SM-algorithm. The broadcast process of the SM-algorithm consists of T+1 communication phases. In the SM-algorithm, the following notations are used. With $x{:}i$, we denote the value x signed by processor i. Thus, $v{:}j{:}i$ denotes the value v signed by j, and then that value $v{:}j$ signed by i. Processor 0 is the source. Each processor i maintains a set V_i, containing the set of properly signed messages obtained thus far. In [3], the algorithm is defined as follows:

```
Algorithm SM(T), T > 0
Initially, for each i, $V_i = \varnothing$.
(1)   The source signs and sends its value to every processor in the system.
(2)   For each i:
        (a)  If processor i receives a message of the form v:0 from the source and it has not
             received a message before, then:
             (1) it lets $V_i$ equal {v}
             (2) it signs and sends the message v:0:i to every other processor.
```

(b) *If processor i receives a message of the form $v{:}0{:}j_1 \ldots j_k$, and v is not in the set V_i, then*
(1) it adds v to V_i
(2) if $k < m$, then it signs and sends the message $v{:}0{:}j_1{:}\ldots{:}j_k{:}i$ to every processor other than $j_1 \ldots j_k$

(3) *For each i:*
When processor i will receive no more messages, it decides choice(V_i) (where choice is a function on the values in V_i.)

3. Path information

The SM algorithm is based on the assumption that for every correct message *m* received by a correct processor *c*, *c* knows the ***path*** (i.e. the sequence of processors) along which message *m* travelled from the source to *c*. This so-called ***path knowledge assumption*** is needed in the broadcast process, to allow a correct processor to determine for any received correct message *m* the set of processors that did not yet receive *m*, and to relay *m* to these processors. This assumption is also generally used in other ICAs.

In a quasi-asynchronous system, the path knowledge assumption does not automatically hold. In this case, in order to get to know the path along which a message travelled, we should include so-called ***path information*** (i.e. information describing the path along which the message travelled from the source to the current processor) in each message.

Of course, malicious processors may try to undetectably corrupt the path information of messages they relay. By having every correct processor sign the path information of each message, the effects of corruption of this ***authenticated*** path information by malicious processors can be drastically restricted. Because the signatures of correct processors can not be undetectably forged by malicious processors, it is possible to deduce from the authenticated path information of a correct message *m* the set of correct processors that already received *m*.

For the correctness of our *authenticated* self-synchronizing ICAs, it is required that every correct processor *c* knows at least, for every correct message *m* that *c* receives, the set of correct processors along which *m* travelled from the source to *c*. Furthermore, it must be able to group any correct message *m* according to a (correct or faulty) description of the path along which *m* travelled to *c*. These two demands are satisfied by including the above-described authenticated path information in each message.

In the following, we assume that every correct message consists of two parts: the ***message value***, i.e. the information that the sender of the message wanted to communicate, and authenticated path information.

4. Definitions and ICA requirements

In this section, we first give some definitions. Then, we formulate the so-called ***ICA requirements***, i.e. the requirements that are sufficient for an authenticated ICA to guarantee interactive consistency.

4.1. Definitions

In the paper, we use the following definitions:

D1. An ICA is a distributed algorithm, running on different processors. We will refer to the algorithm as a whole as the ***ICA***, and to any sub-algorithm of the ICA running on a certain processor *p* as the ***sub-ICA of processor p***.

D2. A ***path-correct message*** is a message with correct path information. Correct path information should describe a ***valid path***, i.e. a path of length less than or equal to 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.

D3. A ***path-faulty message*** is a message with mutilated path information, or path information describing an invalid path.

D4. A ***correct message*** is a correctly authenticated and path-correct message.

D5. A ***faulty message*** is an incorrectly authenticated or path-faulty message.

D6. A message *m* is ***received in time*** in a correct processor *c* if it arrives in *c* in phase *i* ($1 \leq i \leq T+1$) of *c*'s sub-ICA, and the path information of *m* describes a path containing at least i+1 processors.

D7. A message *m* is ***received too late*** in a correct processor *c* if it arrives in *c* after *c* has concluded phase *i* ($1 \leq i \leq T+1$) of its sub-ICA, and the path information of *m* describes a path containing at most i+1 processors.

D8. A message *m*, sent directly from the source to a processor *d* is ***sent in time***, if, after the source started the ICA (by sending the first message of the ICA), *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-ICA 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 processor *j* concluded communication phase *i* of its sub-ICA.

4.2. The ICA requirements

For the message system in the SM algorithm in [3], it is assumed that the following requirements are fulfilled:

MR1. Any correct message which is sent directly from the source s to a processor c arrives in communication phase 1 of the ICA in processor c

MR2. Any correct message which is sent in communication phase i of the ICA from a processor p to a processor q, arrives in communication phase i+1 of the ICA in processor q.

MR3. The absence of messages can be detected.

MR4. For every correct message m received by a correct processor c, c knows the path along which message m travelled from the source to c.

In a synchronous system, where all processors start the protocol simultaneously, these assumptions can be easily satisfied. The communication phases may be synchronized in such a system, i.e. each communication phase begins and ends simultaneously in each processor (either correct or faulty). For this reason, in the requirements above, it is justified to speak of communication phase i of the ICA, without specifying in which processor it takes place, since the definition of communication phase i is equal for all processors.

In our self-synchronizing ICAs, processors do not a priori know when an ICA starts. Therefore, we can not guarantee that each communication phase begins and ends simultaneously in each processor. All processors may start and end any communication phase i at different times. Therefore, for every i and every pair of *correct* processors p and q, we clearly distinguish between communication phase i in processor p and communication phase i in q. Since clocks of faulty processors may run at arbitrary rates, we do not pose any requirement on the length or the presence of communication phases in *faulty* processors.

We will now formulate the ***ICA requirements***, i.e. the requirements which are sufficient for an authenticated ICA to guarantee interactive consistency in a quasi-asynchronous system:

R1. Any correct message which is sent directly from the source s to a correct processor c, and which is sent in time by s to c, arrives in communication phase 1 of c's sub-ICA in processor c

R2. Any correct message which is sent by a correct processor p in communication phase i from p's sub-ICA to a correct processor q, arrives in processor q in or before communication phase i+1 of q's sub-ICA.

R3. In every correct processor c, every communication phase of c's sub-ICA has an a priori known length.

R4. For every correct message m received by a correct processor c, c knows the path information of message m.

In Section 5, we will present our authenticated self-synchronizing ICAs, and prove that these ICAs satisfy the ICA requirements. We know that by satisfaction of the requirements MR1 through MR4 interactive consistency can be guaranteed. Although the ICA requirements are slightly weaker than the requirements MR1 through MR4 stated above, we now show that interactive consistency can also be guaranteed by an authenticated ICA, if it satisfies the ICA requirements (instead of the requirements MR1 through MR4). The main difference between the requirements MR1 through MR4 and the ICA requirements is that in the latter, messages may be delivered out of order (see R2), and any correct processor c only knows the path information of a correct message instead of the path along which it was sent from the source to c. However, we already discussed that knowing the path information is sufficient to group the messages in the decision-making process and to determine the set of correct processors that did not yet receive the message. So, interactive consistency can be guaranteed by an authenticated ICA, if this ICA satisfies the ICA requirements.

5. A description of an authenticated self-synchronizing ICA

In this section, we describe the general idea of an authenticated self-synchronizing ICA. In Section 6, we give an overview of all required assumptions. In Section 7, we prove that the ICA described below satisfies the ICA requirements formulated above.

An authenticated self-synchronizing ICA can be described as follows:

1. *The source s signs and sends a message to every processor i in the system.*

For each processor i:

2. *If processor i timely receives a correct message m, and it has not received a message with path information describing the same path before, i accepts m. If m is the first correct message that i receives, i starts the first communication phase of its sub-ICA 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-ICA ends. (Processor i is able to do this, because i knows a priori the length of every communication phase of its sub-ICA).*

3. *If i receives a faulty message, a correct message that is too late or a correct message with path information describing a path that is identical to a path described in path information of a message that i received before, 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 that is not in the path* *information, i signs message m and i relays m to j, at the end of the communication phase (of i's sub-ICA) in which i received m.*

5. *After the broadcast process of its sub-ICA has ended, i decides on basis of the message values of the messages it has accepted.*

6. Assumptions for self-synchronizing ICAs

The required assumptions 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.

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

$$\frac{1}{1+\rho} \leq \frac{c(t_1) - c(t_2)}{t_1 - t_2} \leq (1+\rho)$$

Thus, we assume that the processor clocks of correct processors are ρ-bounded. Notice that the clock value $c(t)$ of a correct processor's clock may deviate arbitrarily from real-time t.

A3. A processor may concurrently execute different ICAs. We assume that messages of different ICAs can be distinguished, such that every correct processor can determine when it receives the first correct message of a new ICA.

A4. If the source is correct, in every destination processor d, a correct 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 ICA.

A5. In every correct processor c, the first communication phase of c's sub-ICA starts as soon as c receives the first correct message m of this ICA. Message m is either a message from the source sent directly to c, or a message from the source sent to c via a number of intermediate processors.

A6. Every processor in the system may have a different view of time and of the start and end of communication phases in the ICA. For all i with $1 \leq i \leq T+1$, for any correct processor p, let L_i be the length of communication phase i, as viewed from p. Then:

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

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

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

A7. We assume that the processors communicate by means of a reliable point-to-point communication 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 [7].

A8. A correct processor rejects all faulty messages it receives.

A9. Correct messages received in time in a correct processor c are accepted by c.

A10. Correct messages received too late in a correct processor c are rejected by c.

A11. Any correct message that is timely received by a correct processor is accepted and relayed at the end of the communication phase in which it was received.

A12. Any correct message m that is timely received by a correct processor is relayed to all processors that are not in the path information of m. This implies that m is sent to all correct processors that did not yet receive it.

A13. 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.

A14. We assume that every correct message m contains authenticated path information. ❑

7. Proofs

Assume an authenticated deterministic ICA is performed in a fully-connected quasi-asynchronous system S consisting of N processors (including a source s) of which at most T processors behave maliciously. System S satisfies assumptions A1 through A14. We prove that, in S, the ICA requirements are implied by assumptions A1 through A14.

Since, by A6, every correct processor knows a priori the length of each communication phase of its own sub-ICA, requirement R3 is satisfied. Requirement R4 can be satisfied by including authenticated path information in every correct message (A14). Furthermore, in Theorem 1, we prove that all correct messages sent directly from the source s to a correct processor c, which are sent in time by s, are accepted by c (requirement R1). In Theorem 2, we prove that any correct message m which is timely received and accepted by a correct processor p in communication phase i ($1 \leq i \leq T$) of p 's sub-ICA and which is relayed to a correct processor q, arrives in processor q in or before communication phase i+1 of q 's sub-ICA (requirement R2). Finally, in Theorem 3, we prove that any correct message accepted in a correct processor during an ICA, is timely received and accepted in all correct processors before the end of this ICA.

LEMMA 1.1

If a correct processor c receives the first correct message m of an ICA with path information describing a path of n processors, c knows that the ICA has been going on for at least τ_{min}.

Proof:

Assume a correct processor c receives the first correct message m of an ICA with path information describing a path of n processors. Then, c knows that the ICA has been going on for at least τ_{min}. Although it is very well possible that the ICA is going on for a longer time, c can by no means conclude that. For this reason, at receipt of the first correct message of an ICA, any correct processor should always start its sub-ICA in the ***first*** communication phase, in order to be sure that it accepts all messages that may be in time.

(Notice that it is not necessary to require that all correct messages that are accepted were indeed sent in time. Only if the source is faulty, acceptance of a correct message that is not sent in time may influence the decisions taken in the correct processors. However, in this case, by Theorem 3, the decisions in correct processors will still be equal, thus, the interactive consistency conditions are still satisfied.)

We will show that for any n ($2 \leq n \leq T+1$) there always exists a situation in which the ICA has only been going on for τ_{min} at the moment that c receives a message with path information containing n processors.

For n=2, this can easily be seen. In this case, message m was sent directly from the source to c. Any message from a correct source may arrive τ_{min} after the source started the ICA (by A4). For $3 \leq n \leq T+1$, this lower bound τ_{min} is reached, if there is collusion between faulty processors. Then, a faulty processor may sign m with the signatures of n faulty processors (starting with the signature of the source), and send it to c within a minimum real-time interval of length τ_{min}. This proves Lemma 1.1. ❑

THEOREM 1

All correct messages sent directly from the source s to a correct processor c, which are sent in time by s to c are accepted by c.

Proof:

We prove the equivalent theorem:

All correct messages sent directly from the source s to a correct processor c, which are rejected by c, can not have been sent in time by s.

Assume a correct processor c receives the first correct message m of an ICA with path information containing n processors. Then, c knows that an ICA has been going on for at least τ_{min} (by Lemma 1.1).

Now, we distinguish between two cases and show that in both cases the theorem holds.

If $n = 2$ (i.e. m is sent directly from s to c) then, m arrives in c in communication phase 1 of c 's sub-ICA and the path information of m describes a path containing 2 processors. Thus, m has arrived in time in c, and, by A9, m is accepted by c.

If $n > 2$ (i.e. m is not sent directly from s to c, or s is faulty), then c 's first communication phase has a remaining length of $(\tau_{max} - \tau_{min})\cdot(1+\rho)$. Any correct message m ', sent from s directly to c, and arriving within $(\tau_{max} - \tau_{min})\cdot(1+\rho)$ after m, arrives in c in communication phase 1 of c 's sub-ICA and the path information of m ' describes a path containing 2 processors. Thus, m ' has arrived in time in c and, by A9, it is accepted by c. Any correct message m ", sent from s directly to c and arriving in c more than $(\tau_{max} - \tau_{min})\cdot(1+\rho)$ after m, arrives after c has concluded communication phase 1 of its sub-ICA and the path information of m " contains 2 processors. Thus, m " is received too late and, by A10, m " is rejected by c. However, c is sure that m " arrives more than τ_{max} after m was sent by s (even if c's clock runs a factor $(1+\rho)$ faster than real-time, then, after receipt of m, a real-time interval of at least $\tau_{max} - \tau_{min}$ has elapsed), and thus, m " can not have been sent in time by s.

So, any correct message that is rejected by c can not have been sent in time by s. This proves Theorem 1. ❑

LEMMA 2.1

Once a correct processor p has concluded the first communication phase of its sub-ICA, then, within a real-time interval of $(\tau_{max} - \tau_{min})\cdot(1+\rho)^2 + \tau_{max}$, all other correct processors have also concluded the first communication phase of their sub-ICA.

Proof:

Assume a correct processor p receives the first correct message m of an ICA. From A11, we may conclude that all correct processors that already received m before p received it, have already concluded the first communication phase of their sub-ICA. Processor p sends m to the other correct processors that did not yet receive m, at the end of p's first communication phase (by A11 and A12). In real-time, m arrives in any correct processor c at or before τ_{max} after p sent it.

If for any correct processor c, m is the first correct message that c receives, then, viewed from c, c 's first communication phase has a remaining length of $(\tau_{max}-\tau_{min})\cdot(1+\rho)$. Since c 's processor clock may run up to a factor of $(1+\rho)$ slower than real-time, in real-time, c's first communication phase may take up to a factor $(1+\rho)$ longer to complete. Thus, in real-time, c's first communication phase has a remaining length less than or equal to $(\tau_{max} - \tau_{min})\cdot(1+\rho)^2$.

If for this correct processor c, m is not the first correct message that c receives, then it must have received the first correct message m' at or before the moment it received m. Assume that m ' is has path information describing a path containing n ' processors. Then, at receipt of m ', viewed from c, c 's first communication phase has a remaining length of $(\tau_{max} - \tau_{min})\cdot(1+\rho)$. Since c's processor clock may run up to a factor of $(1+\rho)$ slower than real-time, in real-time, c's first communication phase may take up to a factor $(1+\rho)$ longer to complete. Since m ' arrives in p at or before the moment that m arrives in p, at receipt of m, in real-time, c's first communication phase has a remaining length less than or equal to $(\tau_{max} - \tau_{min})\cdot(1+\rho)^2$.

In both cases, for any correct processor c, the first correct message is received at or before τ_{max} after p concluded the first communication phase of its sub-ICA. Furthermore, in real-time, for any correct processor c, at receipt of a message from p, the remaining length of the first communication phase of c is less than or equal to $(\tau_{max} - \tau_{min})\cdot(1+\rho)^2$. Thus, in real-time, within a time span of $(\tau_{max} - \tau_{min})\cdot(1+\rho)^2 + \tau_{max}$ after p has concluded its first communication phase, all other correct processors have also concluded the first communication phase of their sub-ICA. ❑

LEMMA 2.2

After communication phase i $(1 \le i \le T)$, in real-time, all correct processors are synchronized within a real-time bound of $(L_{i+1} / (1+\rho)) - \tau_{max}$.

Proof:

We will prove this lemma by induction on i.

Basis i =1

From Lemma 2.1 we know that after communication phase 1, all correct processors have been synchronized within a real-time bound of $(\tau_{max} - \tau_{min})\cdot(1+\rho)^2 + \tau_{max}$, which is equal to $(L_2 / (1+\rho)) - \tau_{max}$.

Induction step (i >1)

We assume that Lemma 2.2 holds for $j = i - 1$. Thus, the induction hypothesis is:

After communication phase i - 1(2 ≤ i ≤ T), in real-time, all correct processors are synchronized within a real-time bound of $(L_i / (1+\rho)) - \tau_{max}$.

Now, we prove that Lemma 2.2 also holds for $j = i$.

Notice that the maximal real-time difference in synchronization occurs between a correct processor s which runs a factor $(1+\rho)$ slower than real-time and a correct processor f which runs a factor $(1+\rho)$ faster than real-time.

For all i with $1 \le i \le T+1$ and any processor p, let $L_i(p)$ be the real-time length of communication phase i in p. For a correct processor s, which runs a factor $(1+\rho)$ slower than real-time, the real-time length of communication phase i in s is $L_i(s) = L_i\cdot(1+\rho)$. For a correct processor f, which runs a factor $(1+\rho)$ faster than real-time, the real-time length of communication phase i in f is $L_i(f) = L_i/(1+\rho)$. So the real-time difference in synchronization may increase by an amount less than or equal to $L_i(s) - L_i(f)$. From the induction hypothesis, we know that the real-time difference after communication phase i-1 was less than or equal to $(L_i / (1+\rho)) - \tau_{max}$, which is equal to $L_i(f) - \tau_{max}$. After communication phase i, the maximal real-time difference in synchronization is $(L_i(f) - \tau_{max}) + (L_i(s) - L_i(f)) = L_i(s) - \tau_{max}$, which is equal to $L_i\cdot(1+\rho) - \tau_{max}$. Since, by A7, for $i > 1$, $L_{i+1} = L_i\cdot(1+\rho)^2$, the maximal real-time difference in synchronization after communication phase i is equal to $(L_{i+1} / (1+\rho)) - \tau_{max}$. This proves Lemma 2.2. ❑

THEOREM 2

Any correct message m which is timely received and accepted by a correct processor p in communication phase i $(1 \le i \le T)$ of p 's sub-ICA and which is relayed to a correct processor q, arrives in processor q in or before communication phase i+1 of q 's sub-ICA.

Proof:

Assume we have two correct processors p and q. Assume furthermore that p receives in communication phase i of its sub-ICA $(1 \le i \le T)$ a correct message m. Assume that q did not yet receive m (This is possible, since m may have travelled via faulty processors only). Then, at the end of communication phase i of p 's sub-ICA, at a certain time t, p relays m to q. Then, m arrives in q at or before $t + \tau_{max}$.

According to lemma 2.2, at the end of communication phase i $(1 \le i \le T)$, all correct processors are synchronized within a real-time bound B, with $B = (L_{i+1} / (1+\rho)) - \tau_{max}$. Since p concludes communication phase i of its sub-ICA at time t, we may conclude that the earliest time at which q concludes communication phase i of its sub-ICA is at real-time $t - B$. We must prove that m always arrives in q in or before communication phase i+1 of q 's sub-ICA. Processor q concludes communication phase i+1 of its sub-ICA at or after $t + L_{i+1}(q) - B$. From A6, we know that, viewed from q, communication phase i+1 has a length of L_{i+1}. In real-time, this communication phase may be up to a factor $(1+\rho)$ longer or shorter, depending on whether q's processor clock runs slower or faster than real-time. Thus, in real-time, the minimum length of communication phase i+1 for q is $L_{i+1}(q) = (L_{i+1} / (1+\rho))$. Processor q concludes communication phase i+1 of its sub-ICA at or after $t + L_{i+1}(q) - B$, which is equal to $t + \tau_{max}$.

Thus, m always arrives in q in or before communication phase i+1 of q 's sub-ICA. ❑

THEOREM 3

Any correct message accepted in a correct processor during an ICA, is timely received and accepted in all correct processors before the end of this ICA.

Proof:

By A9 and A10, correct messages are only accepted in a correct processor, if they arrive in time in that processor.

Assume that a correct message *m* is timely received in a correct processor *c* during an ICA. Then we must prove that *m* is timely received in every correct processor before the end of the ICA.

Message *m* is received in processor *c* in a certain communication phase *i* of *c* 's sub-ICA, with $1 \le i \le T+1$. We distinguish between two cases:

Case 1: $1 \le i \le T$

Let *d* be a correct processor in the system ($d \neq c$). We must prove that *m* is timely received in *d*. If *d* is in the path information of *m*, then *d* has already timely received and accepted *m*. If *d* is not in the path information of *m*, then at the end of communication phase *i* of *c* 's sub-ICA, *c* will relay *m* to *d* (by A12). Then, from Theorem 2, we know that *m* arrives in *d* in or before communication phase i+1 of *d* 's sub-ICA. Since *c* is correct and *m* is timely received in *c*, and $i \le T$, *m* will also be timely received and accepted in *d*. Thus, *m* is always timely received in *d*.

Case 2: $i = T + 1$

We must prove that any correct message *m* timely received in communication phase T+1 of *c* 's sub-ICA, must already have been timely received and accepted in all other correct processors. If the source is correct, then *m* is timely received and accepted by all correct processors in the first communication phase of their sub-ICA. Assume that the source is faulty. Message *m* must have path information containing at least T+2 processors (including the source). Since at most *T* of these processors are faulty, *m* must have been accepted and relayed by at least one correct processor *d*. This processor *d* must have timely received message *m* before phase T+1 of its sub-ICA. Thus, for a certain *j*, with $1 \le j \le T$, *d* has timely received and accepted *m* in communication phase *j* of its sub-ICA. From A12 and Theorem 2, we know that all correct processors must have timely received and accepted message *m* at or before communication phase j+1 of their sub-ICA. Since $j \le T$, all correct processors have timely received and accepted message *m* at or before communication phase T+1 of their sub-ICA.

Thus, any correct message accepted in a correct processor during an ICA, is timely received and accepted in all correct processors before the end of this ICA. ❑

8. Adapting existing ICAs

The adaptations that must be made to existing authenticated deterministic ICAs such that they satisfy the ICA requirements in a quasi-asynchronous system, are:

1. A correct processor rejects all faulty messages it receives.
2. Correct messages are accepted by a correct processor, provided they arrive in time in that processor, otherwise they are rejected.
3. In every correct message *m*, authenticated path information is included, describing the path along which *m* has travelled from the source to the processor that receives *m*.
4. Any message *m* that is accepted by a correct processor *c* is relayed to all processors that are not in *m*'s path information, at the end of the communication phase in which *m* arrives in *c*.

9. Conclusion

Most ICAs are based on the assumption that all correct processors start the algorithm simultaneously. It is hard to satisfy this assumption in a distributed system. In this paper, we have described a new class of self-synchronizing ICAs that guarantee interactive consistency *without* the above-mentioned assumption.

10. 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., Shostak, R. and Pease, M., The Byzantine Generals Problem, in: **ACM Transactions on Programming Languages and Systems**, Vol.4, No.3, July 1982, pp. 382-401.

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

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

[6] 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.

[7] 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.

[8] Coan, B.A., et al., The Distributed Firing Squad Problem, in: **The 17th ACM Symposium on the Theory of Computing**, ACM, New York, 1985, pp.335-345.

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