A posteriori AGREEMENT
FOR CLOCK SYNCHRONIZATION
ON BROADCAST NETWORKS

INESC Technical Report RT/62-92

L. Rodrigues, P. Veríssimo

January 1992

LIMITED DISTRIBUTION NOTICE

A shorter version of this report was published in the Digest of Papers, The 22th International Symposium on Fault-Tolerant Computing, July, 1992, Boston - USA,

A posteriori agreement for clock synchronization on broadcast networks*

Luís Rodrigues, Paulo Veríssimo
Technical University of Lisboa
INESC†
e-mail:...ler@inesc.pt, paulov@inesc.pt

Abstract

We present a clock synchronization algorithm, dubbed *a posteriori agreement*, based on a new variant of the well-known convergence non-averaging technique. By exploiting an obvious characteristic of broadcast networks, the effect of message delivery delay variance is largely reduced. In consequence, the precision achieved by the algorithm is drastically improved. Accuracy preservation is near to optimal. Our solution, however, does not require the use of dedicated hardware.

*A shorter version of this report was published in the Digest of Papers, The 22th International Symposium on Fault-Tolerant Computing, July, 1992, Boston - USA, © 1992 IEEE

†Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 - 6º - 1000 Lisboa - Portugal, Tel.+351-1-3100000 This work has been supported in part by the CEC, through Esprit Project 1226 - DELTA-4, and JNICT, through Programa Ciência.

Contents

1 Introduction

A global timebase is a mandatory requirement of distributed real-time systems, to allow decentralized agreement on the time to trigger actions, or on the time at which events occurred. It is also a very useful block for building fault-tolerant distributed algorithms. In consequence, the availability of global time in distributed systems, namely fault-tolerant real-time ones, should be encouraged.

It is possible to provide such a timebase by using a centralized time service, resident in a single node of the system. This solution is not fault-tolerant, exhibits poor performance if clocks need to be frequently read, and errors are introduced due to variation of transmission delays. The common solution for the global timebase problem lies on using the processor hardware clock to create a virtual clock at each node, which is locally read. All virtual clocks are synchronized by a *clock synchronization algorithm.* Surveys of existing clock synchronization algorithms can be found in [20,18,12].

The context of the present work was the design of a global timebase for the real-time and fault-tolerant distributed DELTA-4[1] architecture [17] based on broadcast LANs (local area networks). The result was a new variant of the convergence non-averaging technique. The algorithm, tolerant of crash and omission faults in general and arbitrary clock faults, is highly precise and accurate, without any hardware support.

The algorithm uses properties of broadcast networks to drastically attenuate the traditional limitation imposed by message delivery delay variance on the obtained precision. However, it preserves close-to-optimal rate drift. To our knowledge, this presents a significant improvement over previous software-based algorithms. We also know of no solution for the clock synchronization problem that fully exploits the intrinsic attributes of broadcast networks.

The protocol works as follows: synchronization starts with each processor disseminating a start message at a pre-agreed instant on its clock; after a series of broadcast exchanges, each tentatively initiating a new virtual clock, an agreement is obtained both on a broadcast yielding high precision, and on the clock to synchronize from in order to yield the best accuracy possible. It was thus dubbed **a posteriori agreement**.

The paper is organized as follows. The main concepts about clock synchronization, needed along the text, are briefly introduced in section 2. The approach taken is described in section 3. In introducing our solution, section 4 describes a broadcast network model, while section 5 presents the concept of *a posteriori agreement* and describes the synchronization protocol. Concluding remarks are are provided in section 6 and formal proof of correctness is presented in Appendix.

[1] Delta-4, ended in December 1991, was a CEC Esprit II consortium, formed by Ferranti-CSL (GB), Bull (F), Credit Agricole (F), IEI (I), IITB (D), INESC (P), LAAS (F), LGI (F), MARI (GB), NCSR (GB), Renault (F), SEMA (F), Un. of Newcastle (GB), designing an open, dependable, distributed architecture.

2 The clock synchronization problem

The goal of clock synchronization is to establish a global timebase in a distributed system composed of a set of processes which can interact exclusively by message exchange. Processes can only observe time through a *clock*. For convenience, a clock is usually represented by a function $c(t)$ that maps (non-observable) real time[2] to *clock time* (notation closely follows that of [20]).

If every process had access to a common reference time, it could use it as the global timebase. One way to grant such an access is through a radio receiver, capturing an international time standard like the *Universal Time Coordinated, UTC*. However, this approach can be economically very expensive, thus only a limited number of perfect clocks may be made available in the system. Global time can then be obtained trough remote readings of one of the reliable time sources. This approach also possesses several drawbacks. Since processors communicate only through message exchange, clock reading can be very expensive in traffic and time. Moreover remote interactions introduce reading errors due to unpredictable message transit delays and some form of agreement protocol should be implemented if the reading is also pretended to be fault-tolerant. Furthermore, the availability of radio time may be insufficient for some applications [12].

Another common solution to achieve a global timebase is to provide each processor in the distributed system with an imperfect physical clock pc. The clock at a correct processor k can then be viewed as implementing, in hardware, an increasing, continuous[3] function pc_k that maps real time t to a clock time $pc_k(t)$ which, for some positive constants μ_p and ρ_p, satisfies:

PC 1 (Physical Clock Initial value)

$$0 \leq pc_k(0) \leq \mu_p$$

PC 2 (Correct Physical Clock Rate)

$$0 \leq 1 - \rho_p \leq \frac{pc_k(t_2) - pc_k(t_1)}{t_2 - t_1} \leq 1 + \rho_p \quad for \;\; 0 \leq t_1 < t_2$$

Through a clock synchronization algorithm it is possible to derive, from the physical clock at each node k, a virtual clock vc_k satisfying the following conditions:

VC 1 (Precision)

$$|vc_k(t) - vc_l(t)| \leq \delta_v, \quad for \;\; 0 \leq t$$

[2] In an assumed Newtonian time frame.

[3] It is known that digital clocks have a finite granularity and increase by ticks. This notion is of utmost relevance to derive ordering and synchronism properties of real-time systems [12], i.e. in using clocks. In the definition of the clock proper and of a synchronization algorithm, as in this paper, we chose to simplify our expressions in this matter, using the continuous clock approximation, since the first aim of the paper is to explain the reader a new method to synchronize clocks. We will indeed take granularity into account in a report about implementation of the algorithm and quantification of its properties, that we are currently preparing.

VC 2 (Rate)

$$1 - \rho_v \leq \frac{vc_k(t_2) - vc_k(t_1)}{t_2 - t_1} \leq 1 + \rho_v, \ \ for \ 0 \leq t_1 < t_2$$

VC 3 (Envelope Rate)

$$1 - \rho_\alpha \leq \frac{vc_k(t) - vc_k(0)}{t} \leq 1 + \rho_\alpha, \ \ for \ 0 \leq t$$

VC 4 (Accuracy)

$$|vc_k(t) - t| \leq \alpha_v, \ \ for \ 0 \leq t$$

Precision δ_v characterizes how closely virtual clocks are synchronized to each other, ρ_v is the drift rate of virtual clocks, ρ_α is the long term drift rate of virtual clocks, and α_v characterizes how closely virtual clocks are synchronized to real time any moment and this is the correct use for *accuracy* of clocks. Due to the nonzero drift rate of physical clocks, accuracy cannot be ensured unless some external source of real time is available. When an external source of real time is used to provide accuracy, the algorithm is said to implement *external synchronization*. When such source is not used (often it is not available in the system) the algorithm is said to implement *internal synchronization*. In the context of internal synchronization, a good algorithm should maintain clocks as close as possible to the best real time source available, which may be one of the correct clocks in the system. In that sense, of minimizing[4] ρ_v and ρ_α, it should *preserve* accuracy, and that term will be used in this paper when informally discussing these attributes.

Note that in the context of internal synchronization, the term *accuracy* is often used to identify the *envelope rate* property. However, while *accuracy* (as it is defined here) compares the instantaneous clock values with real time, *envelope rate* states that correct logical clocks are within a linear envelope of real time. On the other hand, the *rate* property more restrictive than the *envelope rate* since it bounds the the drift of virtual clocks in any time interval. Also note that the envelope rate can be smaller than rate.

Since physical hardware clocks can be permanently drifting from each other, virtual clocks must be re-synchronized from time to time. A clock synchronization algorithm should then be able to:

- generate a periodic re-synchronization event. The time interval between successive synchronizations is called the re-synchronization interval, denoted T.
- provide each correct processor with a value to adjust the virtual clocks in such a way that conditions **VC 1** and **VC 2** hold.

The clock adjustment can be applied instantaneously or spread over a time interval. In both techniques, for the sake of convenience, the adjustment is usually modeled by the start of a new virtual clock upon each re-synchronization event.

[4] In any case, limited to ρ_p [22].

The computation of the adjustment can be modeled by the evaluation of a *convergence function* [20]. The *precision enhancement property* specifies the best precision guaranteed after any two clock value evaluations at different processors. The worst-case clock precision, δ_v, is obtained by adding the term due to the convergence function to the imprecision generated by the drift between clocks during the re-synchronization interval T. However, since the drift, ρ_p in **PC 2**, is typically of the order of $10^{-6}s$, the precision enhancement property of the convergence function is the relevant factor. It should be noted that the resynchronization interval cannot be made arbitrarily small: resynchronization periods should not overlap and should be long enough to allow adjustments to be spread (furthermore if resynchronization is performed too frequently, the traffic overhead may be significant).

3 The Approach taken

The applications in view for real-time systems in general require accuracy towards some real time reference. When a good external synchronization source is not available, algorithms that preserve accuracy, like the one in [22], become important. Additionally, those systems oriented to distributed computer control require a precision better than that normally achieved with software-based algorithms.

In fact, a major limitation of all known software clock synchronization algorithms designed for arbitrary networks, is that precision is limited either by the variance of the message delivery delay [14], or by its upper bound [22]. This problem may be attenuated in special architectures, either by implementing clock synchronization exclusively by hardware [8,13] or by using hybrid schemes [18,11] which attempt at reducing that variance. Probabilistic or statistical solutions to damp the effect of the variance have also been proposed [4,2]. Both approaches are not without disadvantages: hardware solutions are dedicated, while the traffic overhead of probabilistic solutions increases with the desired probability of success.

An alternative path was followed here, based on the observation that a majority of the distributed systems requiring clocks (eg. real-time) are based on broadcast LANs. In fact, local area networks are commonly in use today. However, we know of no previous solution for the clock synchronization problem that fully exploits the intrinsic attributes of these networks: error rate is low, transmission delay is bounded but with high variance, median transmission delay is close to the minimum, and message reception is *tight* in absence of errors, meaning that the low-level message reception signal occurs at approximately the same time in all nodes that receive it. This is a crucial feature for the mechanism underlying the synchronization algorithm, as will be shown ahead.

Protocols using the convergence-non-averaging technique [7,22,1] are attractive. Since what is disseminated is the event that "a node believes it is now a pre-agreed time" rather than a response to a read clock request, they are inherently resilient to failures, requiring less messages and synchronization cycles than averaging algorithms [14,16,3].

Before proceeding, we present our assumptions about the system:

- **clocks** may have arbitrary failures (eg. provide erroneous or conflicting values when read);

- clock server **processes** (the ones running the protocol over the network) may have failures from crash to uncontrolled omission or timing failures;
- the maximum number of clock-process pairs with failures during a protocol execution is f_p.

The reader will note that the combined assumption of arbitrary-failure clocks and "omissive"-failure processes is realistic and not constraining, by allowing faulty processes to be arbitrarily delayed or even omit their participation in the algorithm in an uncontrolled manner, whereas their "assertive" failed behavior is limited to sending wrong clock values, thus avoiding impersonation, collusion, etc. This removes the difficulty of handling genuinely arbitrary-failure (eg. Byzantine) processes. However, we believe the results presented here can be extended to the latter, by using signatures [22] and redundant broadcast channels.

The network is a single-channel broadcast LAN, as detailed ahead:

- the **network** components are fail-silent, confined to crash if they exceed a given number of omission failures;
- during a protocol execution there is a bound f_o on the number of omission failures produced by the network.

It is possible to put a bound on the time to send a message, to process a received message and read a clock value, etc.:

- both the network and the clock server sub-systems are **synchronous**, in the sense of displaying known and bounded processing and communication delays.

A study on the influence of network timing properties on clock synchronization presented in [12] will help explaining our method. It decomposes a message delivery delay in the following terms:

- *Send Time*, Γ_{send}, to assemble the message and issue the send request;
- *Access Time*, Γ_{access}, for the sender to access the channel;
- *Propagation Time*, Γ_{prp}, for the channel to copy the message to all recipient links[5];
- and *Receive time*, Γ_{rec}, to process the message at the receiver.

The precision of an algorithm is influenced by the variances, $\Delta\Gamma_{...}$, of these terms, which together make up the message delivery delay variance.

We find the algorithm of Srikanth & Toueg appealing for its simplicity and ease of implementation, and because it yields optimal accuracy [22]. One of its problems is that in order to achieve sufficient evidence [20] processes relay messages, which allows the difference between two synchronization actions at different nodes to be as large as

[5] This is the physical propagation time, dependent on the variable distance between nodes.

the maximum message delivery delay. Were it not so, the difference would nevertheless be of the order of the variance in delivery delay. In reasoning about the potential of broadcast networks, two ideas from other algorithms struck our attention. An algorithm by Babaoglu [1] stipulates a property whereby an event is generated in all processes at "the same" real time. That event is simulated by a protocol. In a broadcast network, it is easily achieved by the indication of reception of a message[6]. However, errors may occur and the message may not get to all nodes. An algorithm by Cristian [4], on the other hand, makes several attempts to read time from a server, probabilistically expecting to get a good result value. Let us consider the operation of broadcasting a read command to all clocks in a LAN and get *all* replies in a bounded time, despite errors. In a real-time LAN it is possible to define a bound on the number of tries and the amount of time needed to execute the operation above[7].

The reader will now note four attributes of such a fault-less broadcast which are crucial for the understanding of the algorithm proposed:

- (i) Send and Access errors do not count in a single fault-less broadcast;
- (ii) the message S transmitted arrives virtually at the same time on all nodes, the difference corresponding to the Propagation error;
- (iii) processing times of S reception at any two nodes vary at most by the Receive error;
- (iv) all replies to S get back to the transmitter.

If message S, addressed to all including the sending node, meant: "Let us synchronize! I think the time is H. What time is it on your clocks?", one concludes the following:

- *precision enhancement:* in response to S, a new virtual clock is tentatively initiated everywhere with H, at the same physical time more or less an error equal to the Propagation plus Receive errors;
- *accuracy preservation:* also at that time, the clock of each recipient is read and delivered back to the sender; the sender selects the best clock in terms of accuracy (eg. the median of the clocks, in internal synchronization, or one of those that have a time reference, in external synchronization) and computes its difference to H, to adjust accuracy of the tentative clocks.

This happens every time a process broadcasts S with success. There will be a number of tentative virtual clocks launched, and an agreement protocol is run (a posteriori) to disseminate the chosen one, together with the adjustment, through the clock processes.

Precision obtained at the end of execution will depend not only on the Receive and Propagation errors, but also on the clock rate drift, as with other agreement-based

[6] See section 4 for a formalization of this statement.

[7] See [24] for details.

synchronization protocols. The time required to run the agreement influences precision by a factor of ρ_p. However, its effect can be neglected, given that ρ_p is very small, as discussed in section 2. The Receive error may be practically canceled with co-processors and interrupt treatment, provided that there is an upper bound for message reception interrupt service latency and that bound is small. Then it may be intuitively said that precision is optimal, in the sense that it cannot be better than the variance of the difference between physical reception times of a message at any two nodes (Propagation error).

Accuracy [8], on the other hand, cannot be optimal in the terminology of Srikanth & Toueg, but is very close to it. The worst of the correct clocks will form a bound of the rate drift envelope. The algorithm will, in worst-case, synchronize by that clock, deviating to the outside of the envelope at most by the measure of the maximum Receive error, which as just discussed can be made very small[9]. If a method could be devised to prove that in the presence of a sufficient number of clocks, the median clock would be away from the envelope bounds by more that the reading error, then the rate drift would be optimal.

After introducing the model of network used, the ideas informally sketched above will be formally presented.

4 A Model of Broadcast networks

This section shows that broadcast networks have a number of properties on which clock synchronization may be built, namely the ability to deterministically generate a simultaneous event at all correct processes in the system.

In order not to depend on a particular network, the best approach is to define an *abstract broadcast network*, such that standard local area networks or their variants are represented [9,10,6,15]. The network model of [23] is followed, though modified to be more generic. The abstract network components are: the *channel*, which comprises the passive medium and the interfacing electronics; and the *adapter*, comprising the low-level network protocols, implemented partly in VLSI partly in firmware. The abstract broadcast network appears to the user processes/protocols (namely the clock processes) as a low-level service with a set of properties and an interface [10].

Properties

BNP 1 (Broadcast) *Nodes receiving an uncorrupted message transmission, receive the same message* [11].

[8] Obviously taken in the sense of *accuracy preservation*, which for internal synchronization means following a correct hardware clock, i.e. respecting an envelope rate (cf. §2).

[9] Note that statistically, most executions will held optimal rate drift since for deviation to occur a majority of correct clocks must run at the worst case rate drift, which is has a low probability.

[10] This model fits practically any LAN attachment one may think of, from workstation type on-board VLSI controller to separate controller on multiboard computer. The little added functionality may be achieved by modifying the LAN driver or writing a shell on top of it.

[11] A message is a generic name for a piece of encapsulated information that circulates on the network. It may contain a user-level message

BNP 2 (Error detection) *Nodes detect any corruption done by the network in a locally received message and discard it.*

The network is thus prevented from altering messages, impersonating other senders or delivering conflicting information to different processors on the same broadcast. Existing broadcast networks usually implement cyclic redundancy checks for this purpose.

BNP 3 (Bounded Omissions) *In a network with N nodes, in a known interval, corresponding to a series of M unordered message transmissions, omission failures may occur in at most f_o transmissions.*

This assumption yields a very simple solution to the membership problem as explained in the next section. It is equivalent to expecting that in $f_o + 1$ transmissions, at least one is heard by all nodes. It has a very high coverage in LANs, provided that f_o is well chosen. This assumption is used to stipulate the maximum number of omission failures done by the network components during the synchronization protocol execution and, together with property **BNP refbnp:delay** below, enforce a means for detecting faulty processors without having to introduce a membership protocol. Note that an omission failure may be perceived inconsistently, i.e. a transmission that is not seen by only some (or one) of the recipients.

BNP 4 (Bounded Transmission Delay) *The time between any broadcast send request and the relevant delivery at those nodes that receive the message, is bounded by two known constants $\Gamma^{min} < \Gamma^{max}$.*

The variance in the message delivery delay, $\Delta\Gamma$, is then:

$$\Delta\Gamma = \Gamma^{max} - \Gamma^{min}$$

Securing property **BNP 4** for a network depends on network type and on additional assumptions about its operation, namely that rate and inter-arrival time of message transmission requests are bounded. Existence of the bounds just mentioned allows estimating individual transmission delays in the presence of bounded background loads and queue lengths. In these conditions, Γ^{max} holds for every one of several concurrent transmission requests. For details about enforcing reliable real-time operation on a LAN, the reader is referred to [24].

BNP 5 (Tightness) *Nodes receiving an uncorrupted message transmission, receive it at real time values that differ, at most, by a known small constant $\Delta\Gamma_{tight}$.*

It is important to understand the timing properties of local broadcast networks. The Propagation error is very small: in an Ethernet, for example, the maximum difference between the times of physical reception of a message is less than 20 μs. The Receive error cannot be disregarded: however, $\Delta\Gamma_{rec}$ remains more or less constant and may be improved as discussed earlier in the text. On the contrary, the variance of the Access time, $\Delta\Gamma_{access}$, is hardly controlled and it can have a significant range, strongly depending on variations of the network load and other operating factors (eg. collisions

in Ethernet, token rotation time in a Token-passing LAN). It is the dominant term in message delivery delay variance, $\Delta\Gamma$, and given that:

$$\Delta\Gamma_{tight} = \Delta\Gamma_{prp} + \Delta\Gamma_{rec}$$

a relevant timing property of architectures based on local broadcast networks is formulated the following way:

$$\Delta\Gamma_{tight} \ll \Delta\Gamma$$

An aim of the *a posteriori agreement* technique is to improve precision by making the clock synchronization algorithm depend on $\Delta\Gamma_{tight}$ (instead of $\Delta\Gamma$ or Γ^{max}).

5 A posteriori agreement

The principles of using the *Tightness* property (BNP 5) to allow very precise and accurate clock synchronization were already discussed. In the presence of failures though, incorrect processes/clocks may participate, and broadcasts may be only received by a subset (possibly empty) of the nodes in the system. We define *simultaneous broadcast* as a broadcast that is received by all *correct* processes in the system. The clock synchronization algorithm should then be able to:

- (i) ensure that at least one simultaneous broadcast is generated;
- (ii) ensure that all correct processes choose the same simultaneous broadcast to synchronize their clocks, even when several simultaneous broadcasts are generated;
- (iii) ensure that a simultaneous broadcast is generated often enough to prevent virtual clocks to drift apart more that the desired *precision*;
- (iv) to ensure that a new clock, when it starts to be used, has a value that preserves the desired *envelope rate*.

First, it is described how a simultaneous broadcast can be generated and detected. Then, the achievement of *precision* and of *rate* are discussed. Finally, the initialization and integration of clocks are discussed.

5.1 Generating, detecting and agreeing on a simultaneous broadcast

The protocol is very simply based on having every process perform the same two basic actions: broadcast *once* a "start synchronization" message; and reply (in broadcast) to such messages coming from other processes. This way, modification of failure assumptions only influences the number of processes required to run synchronizations successfully.

With the present assumptions, the presence of $f_o + f_p + 1$ processes in the system is required to generate at least one simultaneous broadcast, given that: each node tries only once; f_p processes may not transmit (eg. process omissions or crashes); and f_o network omissions may occur (**BNP 3**).

Detecting the generation of a simultaneous broadcast is more delicate: it requires feedback from the recipients of the broadcast. Let us assume that each correct recipient broadcasts an acknowledgment message $\langle \text{ack}_b \rangle$ in response to a given broadcast $\langle b \rangle$. For the sake of simplicity, and without loss of generality, we define Γ^{ack} including all delays required to disseminate a reply (including $\Delta\Gamma_{tight}$; the time to process the incoming message and to create and send the acknowledgment; and Γ^{max}). To avoid complicating the algorithm with a group membership management protocol, we use a very simple scheme based on two facts:

- as per **BNP 4**, in absence of failures a correct process, after the reception of a broadcast, should receive an acknowledgment message from every other correct process by Γ^{ack} (real) time (cf. § 4);
- with the help of **BNP 3**, which accounts for actual network omissions, faulty processes can be discovered, if they appear to do more than f_o omission failures[12].

The procedure for detection of a simultaneous broadcast is depicted in figure 1 for a better understanding, although it is embedded in the algorithm of figure 2. Let $\mathcal{P}$ be the set of processes in the system. Let $\mathcal{P}_m^i$ be the set of correct processes in execution i (clock synchronization round i), from process's m point of view (initially $\mathcal{P}_m^i = \text{P}$) (line 10). For each processor m and for each broadcast message $\langle b \rangle$, let $\mathcal{A}_m^b$ include all processes from which an $\langle \text{ack}_b \rangle$ message was received (1.40), and let $\mathcal{F}_m^b$ include those processes from which no acknowledgment has been received within the expected time interval (1.50). Let also $\mathcal{D}_m$ be the set of *detected* simultaneous broadcasts.

A given process p can be considered faulty by a process m if p appears to m as having done more than f_o omissions, i.e. appearing in more than f_o $\mathcal{F}_m^b$ sets. In that case, m withdraws it from its view (line 60). When — because all expected replies did eventually arrive, or because some faulty processes were meanwhile withdrawn from $\mathcal{P}_m^i$ — the sets $\mathcal{P}_m^i$ and $\mathcal{A}_m^b$ match (1.70), broadcast $\langle b \rangle$ is a *simultaneous broadcast*, and is inserted in $\mathcal{D}_m$. The number of processors required to generate and detect a simultaneous broadcast is $(f_p + 1)(f_o + 1)$. It is also the number required to execute the complete synchronization protocol. This latter number can be reduced to $(2f_o + f_p + 1)$ if processes are fail-silent (clocks remaining arbitrary), for $f_o > f_p/2$, or to $(2f_p + 1)$, for $f_o \leq f_p/2$. The proof of these bounds, omitted here for space reasons, can be found in the Appendix.

The mechanism just described does not prevent the generation of several simultaneous clock synchronization events. An agreement protocol must be run afterwards, to select only one broadcast, thence the name of the technique: *a posteriori agreement*. The algorithm does not depend on any particular protocol, as long as agreement is reached in a known bounded time. Fault-tolerant agreement protocols are well-known

[12] Hence the utility of BNP 3 as a "synchronizing" property, without which detection of faulty processes would be impossible in a non-space-redundant network.

For processor m.

10	let $\mathcal{D}_m = \emptyset$; $\mathcal{P}^i_m = \mathcal{P}$;	
20	**case** event **of**	
30	message $\langle b \rangle$ is received:	let $\mathcal{A}^b_m = \mathcal{F}^b_m = \emptyset$;
40	$\langle \text{ack}_b \rangle$ message is received from processor p:	add p to $\mathcal{A}^b_m$;
50	Γ^{ack} (real) time after the reception of $\langle b \rangle$:	set $\mathcal{F}^b_m = \mathcal{P}^i_m - \mathcal{A}^b_m$;
60	$\exists \mathcal{M}, p \ \ s.t. \ \ \#\mathcal{M} > f_o \wedge \forall_{b \in \mathcal{M}}, p \in \mathcal{F}^b_m$:	remove p from $\mathcal{P}^i_m$;
70	$\mathcal{A}^b_m = \mathcal{P}^i_m$:	add b to $\mathcal{D}_m$;
80	**end**; /* case */	

Figure 1: Detecting a simultaneous broadcast

and can be easily found in the literature, although existing *reliable broadcast protocols* for broadcast networks are recommended [23,5]. Given that any simultaneous broadcast will do, agreement may be started immediately after detection of the first simultaneous broadcast.

5.2 Achieving precision

The first phase of the algorithm (figure 2) is very similar to the algorithm of [22]. When $vc^{i-1}_m(t) = iT$, processor m decides to start the synchronization activity for round i, sending a $\langle$start, i, $m\rangle$ message (1.2). Since faulty clocks/processes can send $\langle start \rangle$ messages out of time, the "achievement of sufficient evidence" [20] is necessary, before a message is eligible for a new virtual clock. The criterion of [22] is used: given that f_p clock/process pairs may fail in an untimely manner, a $\langle start \rangle$ message can be considered correct if it has been received at least from $f_p + 1$ distinct processes, out of $2f_p + 1$.

A tentative virtual clock is started upon the reception of every $\langle$start, i, $m\rangle$ message (1.4-6). It is kept running in a *candidate* state. All $\langle start \rangle$ messages are acknowledged by correct processors. Before the achievement of sufficient evidence, start messages are acknowledged by an $\langle ack, i, n \rangle$ (1.10). After, they are acknowledged with a $\langle candidate, i, n \rangle$ message (1.8).

Each processor m monitors all acknowledgments to each message from a processor l that it received. Two different sets, $\mathcal{A}^l_m$ and $\mathcal{C}^l_m$, are used: processors responding with $\langle$ack$\rangle$, in $\mathcal{A}^l_m$ (1.14-15) and processors responding with $\langle$candidate$\rangle$, in $\mathcal{C}^l_m$ (1.12-13). The procedure to update $\mathcal{F}^n_m$, $\mathcal{P}^i_m$ and $\mathcal{D}^i_m$ is in lines 4,6;12-15;16-19;21 ($\mathcal{A}^b_m$ is unfolded in $\mathcal{A}^l_m$ and $\mathcal{C}^l_m$). Note that Γ^{ack} can be measured locally, assuming a worst case rate for the local physical clock, by waiting $(1 + \rho_p)\Gamma^{ack}$ on the local clock (1.16).

A candidate clock launched everywhere in response to a start message by a processor e, is only eligible when at least $f_p + 1$ processors recognize it as such (1.20-21), by having replied with $\langle candidate \rangle$ (1.8,1.12). The first processor m detecting the eligibility of e puts it in $\mathcal{D}^i_m$ and the agreement protocol is invoked, to choose the candidate clock to be used during the next re-synchronization round (i) (1.21). Note (1.21) that it is not necessary that all processes see e as candidate ($\mathcal{C}_m \leq \mathcal{P}_m$) but it is necessary that it correspond to a simultaneous broadcast ($\mathcal{C}_m + \mathcal{A}_m = \mathcal{P}_m$).

Recapitulating, each simultaneous broadcast starts a set of candidate clocks that,

For every processor m.

01 **case** event **of**
02 $vc_m^{i-1}(t) = iT$:
03 $\mathcal{P}_m^i = \mathcal{P}$; $\mathcal{D}_m^i = \emptyset$; broadcast ($\langle start,\ i,\ m\rangle$) ;
04 message $\langle start,\ i,\ n\rangle$ received from processor n
05 t_m^n = reception (real) time of $\langle start,\ i,\ n\rangle$
06 $cc_m^{i,n}(t_m^n) = iT$; $\mathcal{C}_m^n = \mathcal{A}_m^n = \mathcal{F}_m^n = \emptyset$;
07 if message $\langle$start, i, $k\rangle$ received from at least $f_p + 1$ distinct processes:
08 broadcast ($\langle candidate,\ i,\ n, vc_m^{i-1}(t_m^n)\rangle$) ;
09 **else**
10 broadcast ($\langle ack,\ i,\ n, vc_m^{i-1}(t_m^n)\rangle$) ;
11 **fi**;
12 message $\langle candidate,\ i,\ l, vc_n^{i-1}(t_n^l)\rangle$ received from processor n:
13 $n \rightarrow \mathcal{C}_m^l$;
14 message $\langle ack,\ i,\ l, vc_n^{i-1}(t_n^l)\rangle$ received from processor n:
15 $n \rightarrow \mathcal{A}_m^l$;
16 $vc_m^{i-1}(t) = vc_m^{i-1}(t_m^n) + (1+\rho_p)\Gamma^{ack}$:
17 $\mathcal{F}_m^n = \mathcal{P}_m^i - \mathcal{C}_m^n - \mathcal{A}_m^n$;
18 $\exists \mathcal{M}, p\ \ s.t. \#\mathcal{M} > f_o \wedge \forall_{b\in\mathcal{M}}, p \in \mathcal{F}_m^b$:
19 $p \leftarrow \mathcal{P}_m^i$;
20 $\exists e\ \ s.t.\ \ \#\mathcal{C}_m^e > f_p \wedge \mathcal{C}_m^e + \mathcal{A}_m^e = \mathcal{P}_m^i$:
21 $\langle start,\ i,\ e, vc_e^{i-1}(t_e^n)\rangle \rightarrow \mathcal{D}_m^i$; start agreement protocol ;
22 $cc^{i,n}$ and $J^{i,n}$agreed :
23 $vc_m^i = cc_m^{i,n} + J^{i,n}$;
24 **end**; /* case */

Figure 2: Achieving precision and rate

due to the Tightness property of the network, are no further apart than $\Delta\Gamma_{tight}$ apart from each other. If a simultaneous broadcast can be detected and agreed in a short amount of real time, these candidate clocks will remain close together and, at the end of the agreement procedure, a new virtual clock satisfying *precision* can start being used. The re-synchronization interval should be chosen long enough to allow a simultaneous broadcast to be generated, detected and agreed but short enough to ensure that virtual clocks do not drift apart more that the desired worst-case precision. A formal proof that the algorithm achieves precision and the inequalities required to parameterize the protocol are presented in Appendix.

5.3 Improving rate

The candidate clocks are initialized with iT (1.6), the value of the sender's virtual clock at the *sending* time of $\langle$start$\rangle$, Γ real time later. Clearly, while this satisfies precision, it does not preserve accuracy.

One could set the candidate clocks to $cc_m^{i,n}(t_m^n) = iT + \Gamma^n$, however, Γ can only be estimated. Instead, note that by reading the clocks in a simultaneous broadcast, we do so at approximately the same time, since $\forall j, k\ |t_j^n - t_k^n| \leq \Delta\Gamma_{tight}$, which is also the time at which the candidate clocks are set to iT (1.6). Then, from the vector of clock readings the differences between clocks and iT at the reading time can be obtained. This allows

the adjustment to be computed, and applied at a (short) later time.

So the candidate clocks can be started with a dummy initial value (say, iT), which will be corrected by an appropriate *adjustment*, $J^{i,n}$, computed by the agreement procedure (l.22-23). It should be noted that the same adjustment must be applied to all clocks, otherwise *precision* will be affected.

Agreement on the adjustment is very simple. All processors disseminate the reception time of the $\langle$start$\rangle$ message, according to their $(i-1)^{th}$ virtual clocks (l.8,l.10). Srikanth and Toueg have shown [22] than no clock synchronization algorithm can achieve a rate drift better than that of the underlying physical clocks. Thus, optimal rate is approached by adjusting $J^{i,n}$ by the physical clock of one of the correct processors. To ensure that a value in the correct envelope of time is chosen, the median clock value should be selected. The agreement protocol should then compute the adjustment $J^{i,n}$ using $J^{i,n} = vc_a^{i-1}(t_a^n) - iT$, a the agreed clock. The algorithm is depicted in figure 2.

During the re-synchronization interval, all virtual clocks drift from real time at the rate of their underlying physical clocks (thus, following the optimal rate). At each re-synchronization, virtual clocks are adjusted by one of the correct virtual clocks. However, there is a window of uncertainty equal to the Tightness interval, $\Delta\Gamma_{tight}$. This is the amount by which virtual clocks can deviate from the optimal real time envelope at every re-synchronization interval. A precise formulation of how much clocks deviate from the optimal real time envelope (essentially $\Delta\Gamma_{tight}$) is presented in appendix. We also show how to transform this succession of virtual clocks in a continuously adjusted clock.

5.4 Initialization and integration

Initialization can be obtained without changing the algorithm. Processors initializing the system send $\langle$start, 0, $n\rangle$ messages. Since no virtual clock is yet started, acknowledgment messages carry a null value in the virtual reception time field, that is, they are in the form $\langle$*candidate,* 0, $n, 0\rangle$ or $\langle$*ack,* 0, $n, 0\rangle$ (needless to say, the computed adjustment $J^{i,n}$ will always be 0). The first virtual clock will then be started as soon as there are enough correct processors in the system.

Integration can be easily implemented through a passive scheme. A processor desiring to join the system does not send any message on the network. It simply waits for the reception of $\langle$*start*$\rangle$ messages from other processors in the system. It then sends acknowledgment messages as specified by the algorithm. Since during the initialization phase, the virtual clock time of the joining process at the reception of the $\langle$*start*$\rangle$ message will be zero, thus being discarded during the agreement protocol. Since a processor might join the system during the re-synchronization period, it might not be able to collect enough information to reach agreement or might agree on a candidate clock that it did not start. In that case the joining process waits for the next re-synchronization period to join the system. Note that it is trivial, as an outcome of the joining process, to include the newcomer in $\mathcal{P}$.

6 Discussion and Conclusions

Convergence-non-averaging algorithms are attractive because they use the convergence function both to generate the re-synchronization event and to adjust virtual clocks. One such algorithm, by Srikanth & Toueg, strongly influenced our design. However, the existing algorithms of this class have a major disadvantage: the precision of their convergence function is directly dependent on the maximum message transit delay in the system. We showed that the properties of broadcast networks can be used to overcome this disadvantage.

The *a posteriori* agreement approach, using the inherent tightness of fault-less broadcasts in local area networks, makes negligible the dependency of precision on message delay variance or maximum. Precision is thus dependent on $\Delta\Gamma_{tight}$. An application of *a posteriori agreement* to arbitrary networks using a probabilistic approach is under study. The bound on number of processes is typical to these algorithms. Note that characterizing network failures separately from processors', besides being necessary to define termination in a non-redundant network, allows to blame each one for the failures done. If they were consolidated, they would not deviate much from the $2f + 1$ bound. As a matter of fact, for $f_o = 1$, we have $2f_p + 2$.

Our solution is effective even under non-negligible loads, provided that the conditions for **BNP 4** still hold. However, it only presents significant advantages when $\Delta\Gamma_{prp}$, $\Delta\Gamma_{rec}$ are small. $\Delta\Gamma_{rec}$ is a matter of capability of the underlying operating system and hardware machinery: some systems may not allow a small and predictable preemption time for reception interrupts. The use of LAN bridges may originate high $\Delta\Gamma_{prp}$ values. In this scenario, *a posteriori agreement* can be used to improve synchronization of nodes in the same segment while global synchronization could be ensured, for instance, by a hierarchical algorithm[21].

Our algorithm, like consistency-based algorithms [14], requires the execution of an agreement protocol. However, the time required to reach agreement has only a second order effect on the achieved precision. We consider the cost in traffic negligible, in face of the usual LAN bandwidths and given the benefit in precision and determinism, with regard to other approaches[4,7,1]. Furthermore, our algorithm does not require any particular agreement protocol (as long as it exhibits bounded termination). Since most fault-tolerant distributed systems are LAN-based and implement some form of agreement protocol, our algorithm can be easily integrated in such architectures.

The hardware-assisted algorithm of Kopetz [11] removes practically all components of the message delay variance, except the propagation error. Our algorithm, without hardware support other than interrupts, only leaves an attenuated receive error besides the propagation error.

With regard to accuracy, note that our algorithm, for the adjustment computation, may choose an external source instead of one of the internal clocks, as long as it has been read simultaneously as well. A radio receiver time (eg. GPSS satelyte receiver) may be used to fulfill this role, limited to just one or a few nodes — to be fault-tolerant — with that capability.

A reliable broadcast protocol suite, with accompanying group membership services, can simplify implementation, and reduce execution time and number of processes. We

have currently an implementation of the protocol over such a service, the DELTA-4 xAMp group communications suite [19].

Acknowledgments

The authors are grateful to A. Casimiro who collaborated in the implementation of *a posteriori agreement* in xAMp. We discussed these ideas with Sam Toueg who provided useful comments concerning the Srikanth & Toueg protocol. Finally, the comments from the referees, which pointed out some inconsistencies and helped improve the readability of the paper, are warmly acknowledged.

A Proof of correctness

A.1 Assumptions

Assumption 1 *There is a known upper bound, Γ_{start}^{max}, on the time required for a $\langle start \rangle$ message to be prepared by a correct process and sent to all correct processes and processed by the recipients of the message.*

Assumption 2 *There is a known upper bound, Γ_{agreem}^{max}, on the time required to elect a candidate clock, after the start of the first candidate clock. For the sake of convenience, the time required to detect the simultaneous broadcast is also included in Γ_{agreem}^{max}.*

Assumption 3 *The difference, in real time, between the starting time of the same candidate clock, $cc^{i,n}$, in two different processors, is bounded by $\Delta\Gamma_{tight}$.*

A.2 Definitions

Definition 1 *Let $ready^i$ be the earliest real time value at which any correct processor n sends a $\langle start,\ i,\ n,\ 1 \rangle$ message, that is when $vc_n^{i-1} = iT$.*

Definition 2 *Let a simultaneous candidate clock be a candidate clock started by all correct processors in the system. That is $cc^{i,n}$ is a candidate clock if and only if $\forall_{k \in \mathcal{P}_c}\ \exists\ cc_k^{i,n}$.*

Definition 3 *Let the candidate clock that is chosen by the agreement protocol as the basis for the virtual clock during the next re-synchronization interval be called the* elected candidate clock. *Let $elected^i$ be the earliest real time value at which any correct processor starts the elected candidate clock $cc_k^{i,n}$*

Definition 4 *Let end^i be the latest real time value at which any correct processor starts the i^{th} virtual clock, cv_k^i.*

The proof, in general, follows that in [22] closely. So, for brevity, we only present here the steps where main differences exits. We first show that the algorithm achieves the precision property.

A.3 Proof of correctness: precision

Lemma 1 *No simultaneous candidate clock is started before a correct processor is ready to do so, that is, $elected^i \geq ready^i$, for $i \geq 1$.*

Proof. No candidate clock is started until at least $f_p + 1$ $\langle start \rangle$ messages are received. Since at least one correct processor must have sent one of these messages, we get $elected^i \geq ready^i$.

Lemma 2 *At the end of the i^{th} re-synchronization period, virtual clocks differ by at most $(1+\rho_p)\Delta\Gamma_{tight}+2\rho_p\Gamma^{max}_{agreem}$. That is, for $i \geq 1$ and for all correct processes n and m, $|vc^i_n(end^i) - vc^i_m(end^i)| \leq (1+\rho_p)\Delta\Gamma_{tight}+2\rho_p\Gamma^{max}_{agreem}$.*

Proof: By definition 1 and assumption 3 the value of a candidate clock, at the moment it is started at any processor, differs from the value of the same clock at any other correct processor by at most $(1+\rho_p)\Delta\Gamma_{tight}$. By Assumption 2, $end^i - elected^i \leq \Gamma^{max}_{agreem}$. Thus, by definition PC 2, $|vc^i_n(end^i) - vc^i_m(end^i)| \leq (1+\rho_p)\Delta\Gamma_{tight}+2\rho_p\Gamma^{max}_{agreem}$.

Assume that the following conditions hold for some $i \geq 1$:

Assumption 4 *By $ready^i$ all correct clocks have already started its $(i-1)^{th}$ virtual clock.*

Assumption 5 *For all correct processes m and n, $|vc_m(ready^i) - vc_n(ready^i)| \leq \delta_{iv}$.*

Lemma 3 *All correct processes start the elected simultaneous candidate clock soon after a correct process is ready to do so. Specifically, $elected^i - ready^i \leq (1-\rho_p)^{-1}\delta_{iv} + \Gamma^{max}_{start}$.*

Proof: The first correct processor to send a $\langle start \rangle$ message, does so at real time $ready^i$. By Assumption 5, the slowest correct clock is no more than δ_{iv} behind. Hence every correct processors sends an $\langle start \rangle$ message no later than $(1-\rho_p)^{-1}\delta_{iv}$ after $ready^i$. By Assumption 1, the $\langle start \rangle$ message will take, at maximum, Γ^{max}_{start} to be disseminated. Thus, the elected simultaneous candidate clock will be started no later than $(1-\rho_p)^{-1}\delta_{iv} + \Gamma^{max}_{start}$ after $ready^i$.

Lemma 4 *All correct processes start their i^{th} virtual clock soon after one correct process is ready to do so. Specifically, $end^i - ready^i \leq (1-\rho_p)^{-1}\delta_{iv} + \Gamma^{max}_{start} + \Gamma^{max}_{agreem}$.*

Proof: By Assumption 1, $end^i - elected^i \leq \Gamma^{max}_{agreem}$. By Lemma 3, $elected^i - ready^i \leq (1-\rho_p)^{-1}\delta_{iv} + \Gamma^{max}_{start}$. Thus $end^i - ready^i \leq (1-\rho_p)^{-1}\delta_{iv} + \Gamma^{max}_{start} + \Gamma^{max}_{agreem}$.

Lemma 5 *The adjustment J^i to virtual clocks is bounded by a know constant, J^{max}, that is $J^{max} \geq |J^i| \ \forall_i$. Specifically, $(1+\rho_p)(\Gamma^{max}_{start}+2\rho_p)+(1-\rho_p)^{-1}\delta_{iv} \geq J^{max} \geq |J^i| \ \forall_i$.*

Proof: By assumption 5, virtual clocks are no more than δ_{iv} apart at $ready^i$. By Lemma 3 it will take at most $(1-\rho_p)^{-1}\delta_{iv}$ until every correct process sends a $\langle start \rangle$ message. Thus, virtual clocks will drift at most $2\rho_p(1+\rho_p)$ in this interval. The fastest clock will increase, at most $(1+\rho_p)\Gamma^{max}_{start}$ during the dissemination of a $\langle start \rangle$ message. The maximum adjustment will then be $(1+\rho_p)(\Gamma^{max}_{start}+2\rho_p)+(1-\rho_p)^{-1}\delta_{iv}$, thus proving the lemma.

Lemma 6 *The period between re-synchronization is bounded. Specifically, $end^i - end^{i-1} \leq (1-\rho_p)^{-1}(T+J^{max}) + \Gamma^{max}_{start} + \Gamma^{max}_{agreem}$.*

Proof: Every correct process n sends the $\langle start \rangle$ message for the $(i-1)^{th}$ re-synchronization when $vc_n^{(i-2)} = (i-1)T$. At the end of the $(i-1)^{th}$ re-synchronization, the virtual clock of any correct processor will exhibit a value greater than $(i-1)T - J^{max}$. So, all correct processors will send their $\langle start \rangle$ messages, before $end^{i-1} + (1-\rho_p)^{-1}(T + J^{max})$. In the worst case end^i will happen $\Gamma_{start}^{max} + \Gamma_{agreem}^{max}$ after all correct processors have sent $\langle start \rangle$, thus proving the Lemma.

We now assume the following two relations:

Assumption 6

$$(T - J^{max}) > (1-\rho_p)^{-1}\Gamma_{agreem}^{max}$$

Assumption 7

$$\delta_{iv} > \underbrace{(1+\rho_p)\Delta\Gamma_{tight} + 2\rho_p\Gamma_{agreem}^{max}}_{\delta_1} + \underbrace{2\rho_p[(1-\rho_p)^{-1}(T + J^{max}) + \Gamma_{start}^{max} + \Gamma_{agreem}^{max}]}_{\delta_2}$$

The term δ_1 corresponds to the result of the convergence function, that is, the real time difference between virtual clocks at the end of the agreement protocol. The term δ_2 corresponds to the worst-case drift during the longest possible re-synchronization interval. Note that assumptions 6 and 7 specify the minimum and maximum values for the re-synchronization period, T, for a given desired precision, δ_{iv}.

Lemma 7 *The maximum deviation between the i^{th} virtual clocks of correct processes m and n is bounded. That is, for $t \in [end^{i-1}, end^i]$, $|vc_n^i(t) - vc_m^i(t)| \leq \delta_{iv}$.*

Proof: By Lemma 2, correct virtual clocks are at most $(1+\rho_p)\Delta\Gamma_{tight} + 2\rho_p\Gamma_{agreem}^{max}$ apart at the end of the i^{th} period. By Lemma 6, $end^i - end^{i-1} \leq (T + J^{max})(1-\rho_p)^{-1} + \Gamma_{start}^{max} + \Gamma_{agreem}^{max}$ and correct clocks drift at the rate $2\rho_p$ in this interval. Thus, $|vc_n^i(t) - vc_m^i(t)| \leq (1+\rho_p)\Delta\Gamma_{tight} + [(T+J^{max})(1-\rho_p)^{-1} + \Gamma_{start}^{max} + \Gamma_{agreem}^{max}]2\rho_p + 2\rho_p\Gamma_{agreem}^{max}$, since δ_{iv} satisfies Assumption 7.

Lemma 8 *Synchronization periods do not overlap. That is, $end^i < ready^{i+1} \leq elected^{i+1}$.*

Proof: The first correct process to send a $\langle start^{i+1} \rangle$ message does so no earlier than $elected^i + (T - J^{max})(1-\rho_p)$. Therefore $ready^{i+1} \geq elected^i + (T - J^{max})(1-\rho_p)$. By Assumption 2, $end^i - elected^i \leq \Gamma_{agreem}^{max}$. Hence, $ready^{i+1} \geq end^i - \Gamma_{agreem}^{max} + (T - J^{max})(1-\rho_p)$. By Lemma 1 $elected^i \geq ready^i$. Thus $end^i < ready^{i+1} \leq elected^{i+1}$, since T satisfies assumption 6.

Lemma 9 *The instantaneous clocks obtained through the algorithm in figure 2 verify the precision property (the proof is by induction on i and will be omitted for brevity).*

A.4 Proof of correctness: envelope rate

We now show that the algorithm achieves *envelope rate* (as above, the proof closely follows that of [22]). The *rate* property will be proven in a further section, after the introduction of continuous clocks. Note that instantaneous clocks exhibit discontinuities introduced by each re-synchronization.

Lemma 10 *For any execution of the algorithm, there exists a constant $\rho_{i\alpha}^{max}$, such that:*

$$\frac{vc_n^i(t) - vc_n^0(0)}{t} \leq 1 + \frac{\rho_p T + (1+\rho_p)\Delta\Gamma_{tight}}{T - (1+\rho_p)\Delta\Gamma_{tight}} \leq 1 + \rho_{i\alpha}^{max}, \quad \forall i \geq 1, t \in [end^i, end^{i+1}]$$

Proof: The complete proof is omitted for brevity. Let $E(t_0)$ be the set of executions of the algorithm in which $ready^1 = t_0$. Consider an execution $e \in E(t_0)$ in which $\forall_{k\geq 1}$, $elected^i = ready^i$, and let the adjustment for the elected clock be $J^{fastest}$. In the execution e the physical clock of process n runs at the maximum possible rate, that is, $1+\rho_p$ with respect to real time. It is clear that execution e is possible. It can be shown that, for process n, the interval of real time between consecutive re-synchronization is $(T - J^{fastest})(1+\rho_p)^{-1}$. In this period its virtual time increases by T. It can also be shown that, for the same process, the maximum adjustment, $J^{fastest}$, will be given by $J^{fastest} = (1+\rho_p)\Delta\Gamma_{tight}$, proving the Lemma.

Lemma 11 *For any execution of the algorithm, there exists a constant $\rho_{i\alpha}^{min}$, such that:*

$$1 - \rho_{i\alpha}^{min} \leq 1 - \frac{(1-\rho_p)\Delta\Gamma_{tight} - \rho_p T}{T + (1-\rho_p)\Delta\Gamma_{tight}} \leq \frac{vc_n^i(t) - vc_n^0(0)}{t}, \quad \forall_{m\in\mathcal{P}_c}, \forall_{i\geq 1}, t \in [end^i, end^{i+1}]$$

Proof: The complete proof is omitted for brevity. Let $F(t_0)$ be the set of executions of the algorithm in which $elected^i = t_o$. Consider an execution $e \in F(t_0)$ where, $\forall_{i\geq 1}$, correct process n, starts the ith clock $\Gamma_{start}^{max} + \Delta\Gamma_{tight}$ in real time after vc_n^{i-1} reads iT. Also, vc_n^i is started at $elected^i + \Delta\Gamma_{tight}, \forall_{i\geq 1}$. In e the physical clock of process n runs at the minimum possible rate with respect to real time. Such an execution is clearly possible. It is easy to show that vc_n^i is a lower bound on the ith virtual clocks of all process in execution e. That is, for $t \in [end^i, end^{i+1}], vc_n^i \leq vc_m^i, \forall_{m\in\mathcal{P}_c, m\neq n}$. For process n, $(1-\rho_p)^{-1}(T - J^{slowest}) + \Gamma_{start}^{max}$ is the interval of real time between consecutive re-synchronizations. In this period, its virtual time increases by T. It can also be shown that, for the same process, the minimum adjustment, $J^{slowest}$, will be $J^{slowest} = (1-\rho_p)(\Gamma_{start}^{max} - \Delta\Gamma_{tight})$, proving the Lemma.

Lemma 12 *The instantaneous clocks of the algorithm in figure 2 verify the envelope rate property, that is $\exists\rho_{i\alpha}$ such that:*

$$1 - \rho_{i\alpha} \leq \frac{vc_k^i(t) - vc_k^0(0)}{t} \leq 1 + \rho_{i\alpha}, \quad \text{for } 0 \leq t_1 < t_2$$

Proof: Choose $\rho_{i\alpha}$ such that $|\rho_{i\alpha}| \geq |\rho_{i\alpha}^{min}|$ and $|\rho_{i\alpha}| \geq |\rho_{i\alpha}^{max}|$. From Lemmas 10 and 11 it easy to show that the algorithm ensures envelope rate.

A.5 Maintaining continuous clocks

Let $t_n^{a,i}$ be the real time when processor n accepts the i^{th} virtual clock. Instantaneous virtual clocks exhibit discontinuities at each re-synchronization since there is usually a nonnull difference between two consecutive clocks, given by: $vc_n^i(t_n^{a,i}) - vc_n^{i-1}(t_n^{a,i})$. Continuous clocks can be obtained this difference is spread over a real time interval Δ_{spread}.

Definition 5 *A continuous virtual clock can be obtained from the instantaneous virtual clocks, using the following function:*

$$vc_m(t) = vc_m^0 \qquad for \quad t < t_k^{a,1}. \tag{1}$$

$$vc_m(t) = vc_m^{i-1} + \frac{(vc_m^1(t_m^{a,i}) - vc_m^{i-1}(t_m^{a,i}))(t - t_m^{a,i})}{\Delta_{spread}} \qquad for \quad t_m^{a,i} < t < t_m^{a,i} + \Delta_{spread}. \tag{2}$$

$$vc_m(t) = vc_m^i \qquad for \quad t_m^{a,i} + \Delta_{spread} < t < t_m^{a,(i+1)}. \tag{3}$$

We now show that the *precision* and the *envelope rate* of the virtual clocks still hold. Additionally, we also prove the *rate* property.

A.5.1 Precision

Lemma 13 *If* $|vc_n^{i-1}(t) - vc_m^{i-1}(t)| \leq \delta(t)$ *then we have* $|vc_n^i(t) - vc_m^{i-1}(t)| \leq \delta(t) + (1 + \rho_p)\Delta\Gamma_{tight}$.

Proof: Let t_n^i be the real time instant at which vc_n^i is started. By the Tightness property , $|t_m^i - t_n^i| \leq \Delta\Gamma_{tight}$, $\forall_{n,m}$. Thus, $|vc_m^{i-1}(t_m^i) - vc_m^{i-1}(t_n^i)| \leq (1 + \rho_p)\Delta\Gamma_{tight}$. At $t_n^{a,i}$, vc_n^i is adjusted accordingly to the virtual clock value of some correct clock, vc_k. The adjustment, is computed using $vc_n^i(t_n^i) = vc_k^{i-1}(t_k^i)$, $t_n^i + \Delta\Gamma_{tight} \leq t_k^i \leq t_n^i + \Delta\Gamma_{tight}$ that is, $|vc_m^i(t_n^i) - vc_n^{i-1}(t_n^i)| \leq \delta(t_n^i) + (1 + \rho_p)\Delta\Gamma_{tight}$. By other words, a new virtual clock vc_n^i is adjusted in such a way that the instantaneous drift from any other previous clock vc_m^{i-1} is bounded by $(1 + \rho_p)\Delta\Gamma_{tight}$. Since $|vc_m^{i-1}(t) - vc_m^i(t)|$ will remain constant thereafter, we will have: $|vc_m^{i-1}(t) - vc_n^i(t)| \leq \delta(t) + (1 + \rho_p)\Delta\Gamma_{tight}$, thus proving the Lemma.

Lemma 14 *The continuous clocks obtained by definition 5 based on instantaneous clocks resulting from algorithm in figure 2 verify the precision property. More precisely:* $|vc_n(t) - vc_m(t)| \leq \delta_{iv} + (1 + \rho_p)\Delta\Gamma_{tight} \leq \delta_v$.

Proof:
For every time value t, exists i such that: (1) $t \in [max(t_m^{a,i-1}, t_n^{a,i-1}),\ min(t_m^{a,i}, t_n^{a,i})]$; or (2) $t \in [min(t_m^{a,i}, t_n^{a,i}),\ max(t_m^{a,i}, t_n^{a,i})]$; (3) $t \in [max(t_m^{a,i}, t_n^{a,i}),\ min(t_m^{a,i+1}, t_n^{a,i+1})]$. By Lemma 9, instantaneous clocks are always within δ_{iv} of each other, thus the proof is trivial for cases (1), and (3). For case (2) the proof follows directly from lemma 13 where we have bounded $|v_m^{i-1}(t) - v_n^i(t)|$ during the interval $[min(t_m^{a,i}, t_n^{a,i}),\ max(t_m^{a,i}, t_n^{a,i})]$.

A.5.2 Envelope rate

Lemma 15 *The continuous clocks obtained by definition 5 based on instantaneous clocks resulting from algorithm in figure 2 verify the envelope rate property, that is* $\exists \rho_\alpha$ *such that:*

$$1 - \rho_\alpha \leq \frac{vc_m(t) - vc_m(0)}{t} \leq 1 + \rho_\alpha, \text{ for } 0 \leq t$$

Proof: Assume $\rho_\alpha \geq \rho_{i\alpha}$. By Lemma 12 the proof is trivial for values of t outside the spreading intervals, $t_m^{a,i} < t < t_m^{a,i} + \Delta_{spread}$. From definition 5, during these intervals, we have: $vc_m^{i-1}(t) \leq vc_m(t) \leq vc_m^i(t)$ for $t_m^{a,i} < t < t_m^{a,i} + \Delta_{spread}$, thus proving the Lemma, for all t.

A.5.3 Rate

Lemma 16 *The difference,* Δ_{vc}*, between two consecutive instantaneous virtual clocks is majored by:*

$$\Delta_{vc} > \delta_{iv} + 2\rho_p[(1-\rho_p)^{-1}\delta_{iv} + \Gamma_{start}^{max} + \Gamma_{agree}^{max}] + (1+\rho_p)\Delta\Gamma_{tight} > |vc_n^1(t_n^{a,i}) - vc_n^{i-1}(t_n^{a,i})|, \quad \forall_{i,n}$$

Proof: By assumption 5, virtual clocks are no more than δ_v apart at $ready^i$. By Lemma 3 the elected simultaneous candidate clock will be started no later than $(1-\rho_p)^{-1}\delta_v + \Gamma_{start}^{max}$ after $ready^i$. Virtual clocks will drift at most $2\rho_p[(1-\rho_p)^{-1}\delta_v + \Gamma_{start}^{max}]$ during this interval. By assumption 3 elected clocks are started within $(1+\rho_p)\Delta_{tight}$ of each other. By assumption 2 it will take at most Γ_{agree}^{max} to agree on the virtual clock. The maximum difference between virtual clocks will then be the sum of these factors.

Lemma 17 *The continuous clocks obtained by definition 5 based on instantaneous clocks resulting from algorithm in figure 2 verify the rate property, that is* $\exists \rho_v$ *such that:*

$$1 - \rho_p - \frac{\Delta_{vc}}{\Delta_{spread}} < \frac{vc_m(t_2) - vc_m(t_1)}{t_2 - t_1} < 1 + \rho_p + \frac{\Delta_{vc}}{\Delta_{spread}}$$

In the time interval between defined by $t_k^{a,i} < t < t_k^{a,i} + \Delta_{spread}$ the continuous virtual clock follows the instantaneous virtual clock, thus its rate equals that of the underlying physical clock, that is ρ_p. During $t_k^{a,i} < t < t_k^{a,i} + \Delta_{spread}$, the difference Δ_{vc} is spread over Δ_{spread}, thus proving the lemma assuming $\rho_v \geq \rho_p + \frac{\Delta_{vc}}{\Delta_{spread}}$.

Lemma 18 *The maximum interval over which the difference between consecutive instantaneous clocks can be spread,* Δ_{spread}*, is given by:* $\Delta_{spread} \leq (T - J^{max})(1+\rho)^{-1}$.

Proof: The spreading interval must be smaller than the minimum real time between any two consecutive re-synchronizations. In the proof of Lemma 10 we have shown that this interval is given by, $(T - J^{max})(1+\rho)^{-1}$, thus proving the Lemma. This gives a minorant for the rate of continuous virtual clocks.

A.6 Conclusion of proof

Theorem 1 *The algorithm of figure 2 is a synchronization algorithm.*

The proof follows directly from definitions 1, 2, 3 and lemmas 14, 15 and 17.

B Proof of bounds on number of processors

In order to generate and detect an universal broadcast, $(f_p + 1)(f_o + 1)$ processors are required.

Proof of necessity: Assume that only $f_o + f_o f_p + f_p$ processors are used. Let $\mathcal{P}^f$ be the set of failed processors and $\mathcal{P}^c$ be the set of correct processors. Consider an execution where f_p processors are failed and do not generate any broadcast. In this execution, only $f_o + f_o f_p$ messages are generated. Consider also that these messages can be grouped in disjoint sets $\mathcal{M}_0$, $\mathcal{M}_1$, ..., $\mathcal{M}_{f_p}$ such that:

$$\forall_{b \in \mathcal{M}_0} \exists_{p_o \in \mathcal{P}^c} \ s.t \ \ p_o \notin \mathcal{A}^b_{p_c} \forall_{p_c \in \mathcal{P}^c} \tag{4}$$

$$\forall_{b \in \mathcal{M}_i} \exists_{p_i \in \mathcal{P}^f} \ s.t \ \ p_i \notin \mathcal{A}^b_{p_c} \forall_{p_c \in \mathcal{P}^c} \tag{5}$$

$$\forall_{i,j}, 1 \leq i \leq f_p, 1 \leq j \leq f_p \ p_i \neq p_j \tag{6}$$

$$\forall_i, o \leq i \leq f_p, \#\mathcal{M}_i = f_o \tag{7}$$

In this execution, messages in $\mathcal{M}_0$ cannot be detected as universal broadcasts due to omissions during the acknowledgment phases. Messages in each set $\mathcal{M}_i$ cannot be detected as universal broadcasts due to lack of acknowledgments from one of the failed processors. Each failed processor, p_i affects a different set of messages, $\mathcal{M}_i$. This is execution is clearly possible. Since failed processors affect disjoint subsets of messages of size f_o they cannot be detected as failed, thus, they are not removed from $\mathcal{P}^c_m$ by any correct processor. It is easy to see that, in this execution, we never have $\mathcal{A}^b_m = \mathcal{P}^c_m \ \ \forall_{m \in \mathcal{P}^c}, \forall_{b \in \mathcal{M}}$.

Proof of sufficiency: Let $\mathcal{M}_o$ be the set of messages affected by omissions. By definition $\#\mathcal{M}_o \leq f_o$. Let $\mathcal{M}_i$ be the set of messages affected by missing acknowledgments from failed processor p_i. If $\#\mathcal{M}_i > f_o$, processor p_i will be removed from $\mathcal{P}^c$. Thus, in order to contribute to prevent a message from being detected as universal broadcast, $\#\mathcal{M}_i \leq f_o$. There are at most f_p failed processors, thus:

$$\#\{\mathcal{M}_o, \mathcal{M}_1, \ldots, \mathcal{M}_{f_p}\} \leq \#\mathcal{M}_o + \#\mathcal{M}_1 + \ldots + \#\mathcal{M}_{f_p} \leq f_o + f_p f_o$$

That is, at most $f_o + f_p f_o$ messages can be disturbed by omissions or failed processors. Thus, its sufficient to generate $f_o + f_p f_o + 1$ messages to detect a universal broadcast. Since failed processors may never generate any broadcast, at most $f_o + f_p f_o + 1 + f_p = (f_o + 1)(f_p + 1)$ processors are required in the system to generate *and* detect a universal broadcast. □

Additionally, there is a lower bound on the number of processes in order for achievement of sufficient evidence to be possible: $2f_p + 1$ [22]. In principle, the maximum of the two should be followed. A simple analysis shows that the previously presented bound is always greater than this one, except for $f_o = 0$, which is non-interesting, so the first bound $(f_o + 1)(f_p + 1)$ is dominant.

If the assumption about process behavior is strengthened to crash failure, maintaining the others (arbitrary clocks, controlled network omissions), the number of processors needed comes down to $2f_o + f_p + 1$. In fact, $2f_o + 1$ messages are needed, since when

processes start doing omissions, they do not recover. Adding f_p which, having failed, may never send their messages, we arrive at the result given. Achievement of sufficient evidence still requires $2f_p + 1$ processors. When $f_o \geq f_p/2$, both expressions are equivalent. Otherwise, the maximum of them should hold.

C Glossary of notation

pc_k physical clock at processor k.

vc_k virtual clock at processor k.

vc_k^i round i instantaneous virtual clock at processor k.

$cc_k^{i,n}$ round i candidate clock at processor k, started upon reception of $\langle \text{start} \rangle$ message from processor n.

$J^{i,n}$ agreed adjustment to the elected candidate clock $cc^{i,n}$.

Δ_{spread} real time interval over each discontinuities in instantaneous virtual clocks are spread to obtain a continuous clock.

ρ_p correct rate of physical clocks.

ρ_v correct rate of virtual clocks.

ρ_α correct envelope rate of virtual clocks.

δ_v precision of virtual clocks.

α_v accuracy of virtual clocks.

T resynchronization interval.

Γ_b network broadcast delay.

$\Delta\Gamma_b$ variability of network broadcast delay.

$\Delta\Gamma_{simul}$ simultaneity of network broadcast reception.

$\langle b \rangle$ broadcast message b.

$\langle \mathbf{ack}_b \rangle$ acknowledgment to broadcast message b.

$\mathcal{P}$ set of processors.

$\mathcal{P}_m^c$ set of correct processors, from processor's m point of view.

$\mathcal{A}_m^b$ set of correct processors, from whom processor m received an acknowledgment to message b.

$\mathcal{F}_m^b$ set of correct processors, from whom processor m did not receive an acknowledgment to message b within a predefined time interval.

f_o network omission faults.

f_p processor faults.

References

[1] Ozalp Babaoğlu and Rogério Drummond. (Almost) No Cost Clock Synchronization. In *Digest of Papers, The 17th International Symposium on Fault-Tolerant Computing*, Pittsburgh, PA, USA, July 1987.

[2] D. Couvet, G. Florin, and S. Natkin. A Statistical Clock Synchronization Algorithm for Anisotropic Networks. In *Proceedings of the Tenth Symposium on Reliable Distributed Systems*, pages 41–51. IEEE, 1991.

[3] F. Cristian, Aghili. H., and R. Strong. Clock Synchronization in the Presence of Omission and Performance Faults. In *Digest of Papers, The 16th International Symposium on Fault-Tolerant Computing*, pages 218–223, Viena - Austria, July 1986. IEEE.

[4] Flaviu Cristian. Probabilistic Clock Synchronization. *Distributed Computing, Springer Verlag*, 1989(3), 1989.

[5] Flaviu Cristian. Synchronous atomic broadcast for redundant broadcast channels. Technical report, IBM Almaden Research Center, San Jose,California, USA, 1990.

[6] FDDI. *FDDI Token-Ring Media Access Control (MAC).* ANSI X3.139, 1987.

[7] Joseph Y. Halpern, Barbara Simons, Ray Strong, and Danny Dolev. Fault-Tolerant Clock Synchronization. In *Proceedings of the 3Rd ACM Symp. on Principles of Distributed Computing*, pages 89–102, Vancouver Canada, August 1984.

[8] A.L. Hopkins, T.B. Smith, and J.H. Lala. FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft. *Proceedings IEEE*, 66(10):1221–1240, October 1978.

[9] *ISO DIS 8802/4-85, Token Passing Bus Access Method*, 1985.

[10] *ISO DP 8802/5-85, Token Ring Access Method*, 1985.

[11] Hermann Kopetz and Wilhelm Ochsenreiter. Clock Syncronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, C-36(8):933–940, August 1987.

[12] Hermann Kopetz and Wolfgang Schwabl. Global time in distributed real-time systems. Technical Report 15/89, Technische Universitat Wien, Wien Austria, October 1989.

[13] C.M Krishna, K.G. Shin, and R.W. Butler. Ensuring Fault Tolerance of Phase-Locked Clocks. *IEEE Transac. Computers*, C-43(8):752–756, August 1985.

[14] L. Lamport and P. Melliar-Smith. Synchronizing Clocks in the Presence of Faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[15] G. LeLann. The 802.3d protocol: A variation of the ieee802.3 standard for real-time lans. Technical report, INRIA, France, 1987.

[16] Jennifer Lundelius and Nancy Lynch. A New Fault-Tolerant Algorithm for Clock Syncronization. In *Proceedings of the 3rd ACM SIGACT-SIGOPS Symp. on Principles of Distrib. Computing*, pages 75–88, Vancouver-Canada, August 1984.

[17] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing.* ESPRIT Research Reports. Springer Verlag, November 1991.

[18] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-Tolerant Clock Synchronization in Distributed Systems. *IEEE,Computer*, pages 33–42, October 1990.

[19] L. Rodrigues and P. Veríssimo. *x*AMp: a Multi-primitive Group Communications Service. Technical Report RT/xx-92, INESC, Lisboa, Portugal, February 1992.

[20] Fred B. Schneider. Understanding Protocols for Byzantine Clock Synchronization. Technical report, Cornell University, Ithaca, New York, August 1987.

[21] K. G. Shin and P. Ramanathan. Clock Synchronization of a Large Multiprocessor System in the Presence of Malicious Faults. *IEEE Trans. Computers*, C-36(1):2–12, January 1987.

[22] T. K. Srikanth and Sam Toueg. Optimal Clock Synchronization. *Journal of the Association for Computing Machinery*, 34(3):627–645, July 1987.

[23] P. Veríssimo and José A. Marques. Reliable broadcast for fault-tolerance on local computer networks. In *Proceedings of the Ninth Symposium on Reliable Distributed Systems*, Huntsville, Alabama-USA, October 1990. IEEE. Also as INESC AR/24-90.

[24] P. Veríssimo, J. Rufino, and L. Rodrigues. Enforcing real-time behaviour of LAN-based protocols. In *Proceedings of the 10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, September 1991. IFAC.