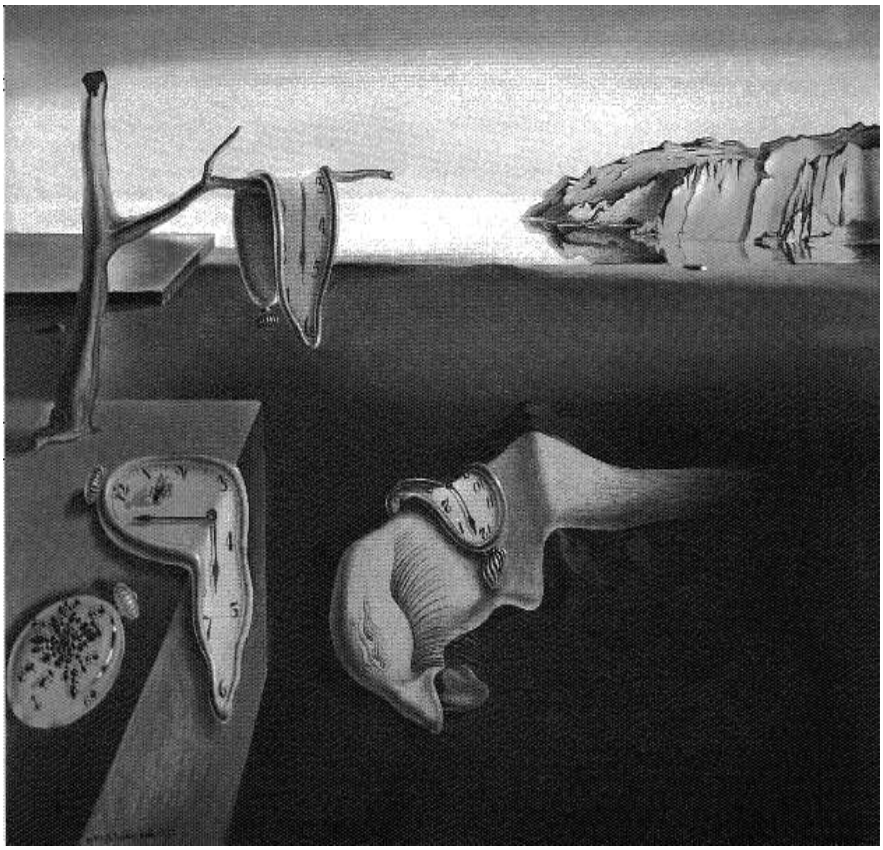
Institut für Automation
Abt. für Automatisierungssysteme

Technische Universität Wien

Projektbericht Nr. 183/1-126
January 2003

Booting Clock Synchronization in Partially Synchronous Systems with Hybrid Node and Link Failures

Josef Widder and Ulrich Schmid



Salvador Dali, "Die Beständigkeit der Erinnerung"

Booting Clock Synchronization in Partially Synchronous Systems with Hybrid Node and Link Failures

JOSEF WIDDER and ULRICH SCHMID*
Technische Universität Wien
Department of Automation
Treitlstrasse 1, A-1040 Vienna
{widder, s}@auto.tuwien.ac.at

Abstract

This paper provides description and analysis a new clock synchronization algorithm for partially synchronous systems with unknown upper and lower bounds on delays. Unlike existing solutions, it relies upon a hybrid failure model incorporating both process and link failures, in both time and value domain, and works during both system startup and normal operation: Whereas bounded precision (= mutual deviation of any two clocks) can always be guaranteed, accuracy (= clocks being within a linear envelope of real-time) and hence progress is only guaranteed when sufficiently many correct processes are eventually up and running.

1 Introduction

Clock synchronization is an important service in distributed systems, see [11] for several application examples. It assumes that every process p owns a discrete clock $C_p(t)$, which is periodically adjusted by a fault-tolerant clock synchronization algorithm. Clock synchronization guarantees that (P) any two correct clocks deviate at most by some D_{max} (*precision requirement*), and (A) that any correct clock remains within a linear envelope of real-time (*accuracy requirement*). Many different clock synchronization algorithms have been proposed in the literature, see e.g. [16, 19, 28] for an overview.

Most of these algorithms assume a synchronous system and cannot handle system startup. In real systems, however, processes start one after the other at unpredictable times, and may not have completed booting when some earlier process starts sending messages. Consequently, during startup, even messages from correct nodes may be lost, and failure assumptions like the one that at most f out of $n \geq 3f + 1$ processes are Byzantine faulty do not hold. Moreover, many real networks (like the Internet) cannot be modeled properly as synchronous systems [2, 8, 32]. To implement clock synchronization[1] in such systems, a time(r)-free startup mechanism is required.

In [34], we provided a solution to this problem that—unlike naive startup algorithms—avoids an increase of the required number of processes: By modifying the clock synchronization algorithm of Srikanth and Toueg [29], which is based upon the well-known consistent broadcast primitive, we provided an algorithm that is complete time- and timer-free and needs only $n \geq 3f + 1$ processes for coping with f Byzantine faulty processes, both during normal operation and system startup. Designed for a partially synchronous system with unknown lower and upper bounds upon delays, it guarantees some precision D_{max}—that depends upon these bounds—during the whole lifetime of the system. Progress of the clocks (linear envelope requirement), however, is only guaranteed when sufficiently many correct nodes are up and running.

Accomplishments: In this paper, we will present and analyze a variant of the algorithm of [34] under (an appropriate extension of) the hybrid *perception-based failure model* introduced in [22]. Making this transition not only provides con-

*This research is part of our W2F-project, which targets a wireline/wireless fieldbus based upon spread-spectrum (CDMA) communications, see *http://www.auto.tuwien.ac.at/Projects/W2F/* for details. W2F is supported by the Austrian START programme Y41-MAT.

[1] Although clock synchronization is traditionally studied in synchronous systems with hardware clocks, it is a useful service in partially synchronous systems with software clocks (counters) as well; see Section 3 for details.

siderably improved fault-tolerance with respect to process failures, but shows that a large number of link failures can be tolerated as well. Our algorithm can therefore be applied even in typical wireless settings, where link failure rates up to 10^{-2} are common. Clock synchronization and, in turn, all algorithms that depend upon synchronized clocks can hence safely be implemented in such loosely coupled systems, despite of unpredictable startup times and communication failures.

Related Work: Although clock synchronization in synchronous systems [3, 14, 16, 19, 27, 28], as well as in partially synchronous systems [4, 15], is a very well-researched field, there are only a few papers [9, 12, 14, 29, 31, 31] that deal with initial synchronization. Rather than considering a full system startup, however, most of those papers are devoted to integrating a new process into an already running system. The only exceptions known to us are [31], which deals with a specific solution to the startup problem in the very specific TTP system architecture, and [9], which considers a timer-based approach for initialization in the MAFT architecture. However, none of those solutions is time(r)-free and works in partially synchronous systems.

Clock synchronization in presence of link failures has been studied in [17] and in part of our previous work [10, 18, 20, 21, 24]. None of those papers considered initial startup, however. Moreover, with the exception of [10, 21], all those papers consider synchronous systems only. In [10], we introduced a time(r)-free implementation of a perfect failure detector in partially synchronous systems based upon consistent broadcasting. Its analysis, which is based upon results from [21], is also based upon the fully-fledged perception-based failure model of [22]. We assumed, however, that all processes are up right from the start. This assumption could now be dropped by the results of the present paper.

Organization of the paper: Section 2 contains an overview of (an extension of) the perception-based failure model of [22]. In Section 3, we provide the hybrid version of the algorithm of [34], along with our major results. Section 4 is devoted to the analysis of our algorithm when insufficiently many processes get up simultaneously, Section 5 shows what happens when the number of processes increases during startup. In Section 7, we analyze our algorithm when sufficiently many nodes are up. The paper is rounded off with some conclusions in Section 8.

2 Perception-based Failure Model

This section contains a very brief overview of our failure model, which slightly extends the perception-based model introduced in [22] by adding messages with history. It consists of an execution model, a basic physical failure model, and a more abstract perception failure model. Both the physical and the perception failure model are *hybrid* ones [1, 33], i.e., distinguish several classes of failures. The advantage of a hybrid failure model is its improved resilience: Less severe failures can usually be handled with fewer processes than more severe ones.

Due to lacking space, we will entirely omit the description of the *physical failure model*. It distinguishes several classes of time and value failures for both processes and links, and uses assertions like "at most ϕ_{av} processes may behave Byzantine in a single round". Due to the exploding number of possible combinations of time and value failures, it is not used for analyzing fault-tolerant algorithms, however. Its primary purpose is the analysis of the assumption coverage in real systems.

The physical failure model can be reduced to a more abstract (and vastly simpler) *perception failure model*, which is similar in spirit to the round-by-round fault detector approach of [6]. It is a generalization of the synchronous model of [25, 26], and is solely based upon the local view (= perception of failures) of every process in the system. The perception failure model is particularly well-suited for analyzing the fault-tolerance properties of distributed algorithms.

2.1 Execution Model

We consider a distributed system of n *processors* connected by a fully or partially connected point-to-point network. All links between processors are bidirectional, consisting of two unidirectional channels that may be hit by failures independently. The system will execute a distributed round-based algorithm made up of one or more concurrent *processes* at every processor. Processes can communicate bidirectionally with each other via the interconnecting links. Every processor is identified by a unique *processor id* $p \in \{1, \ldots, n\}$; every process is uniquely identified system-wide by the tuple (processor id, process name), where the *process name* is chosen from a suitable name space. Since a process will usually only communicate with processes of the same name, we will distinguish processes primarily by their processor ids and suppress process names when they are clear from the context.

Restricting our attention to round-based algorithms, we assume that all processes execute a finite or infinite sequence of consecutive *rounds* $k = 0, 1, \ldots$. In every round except the initial one $k = 0$, which is slightly different, a single process p may broadcast (= successively send) a single message—containing the current *round number* k and a *value* V_p^k depending

upon its local computation—to all processes contained in p's current *receiver set* $\mathcal{R}_p^r \subseteq \{1,\ldots,n\}$.[2] We assume that every (non-faulty) receiver q knows its current *sender set* $\mathcal{S}_q^r = \{p : q \in \mathcal{R}_p^k\}$ containing all the processes that should have sent a message to it, and that a process satisfying $p \in \mathcal{R}_p^k$ (and hence $p \in \mathcal{S}_p^k$) sends a message to itself as well.

Concurrently, for every round number l, process p receives incoming round l messages from its peer processes $\in \mathcal{S}_p^l$ and collects their values in a local array (subsequently called *perception vector*) $\mathcal{V}_p^l = \{V_p^{1,l},\ldots,V_p^{n,l}\}$. Note that $\mathcal{V}_p^l = \mathcal{V}_p^l(t)$ as well as its individual entries $V_p^{i,l} = V_p^{n,l}(t)$ are actually time-dependent; we will usually suppress t, however, in order not to overload our notation. Storing a single value for each peer in the[3] perception vector is sufficient, since any receiver may get at most one round l message from any non-faulty sender. The entry $V_p^{q,l} \in \mathcal{V}_p^l$ (subsequently called *perception*) is either $\emptyset$ if no round l message from process q came in yet (or if $q \notin \mathcal{S}_p^l$), or it contains the received value from the first round l message of process q.

Process p's current round k is eventually terminated at the *round switching time* σ_p^k, which is the real-time when process p switches from round k to the next round $k+1$. At the round switching time, the value $V_p^{k+1} = F_p\big(\mathcal{V}_p^k(\sigma_p^k)\big)$ to be broadcast by process p in the next round $k+1$ is computed as a function F_p of the round k perceptions available in $\mathcal{V}_p^k = \mathcal{V}_p^k(\sigma_p^k)$ at time σ_p^k.

The above execution model is sightly generalized by introducing messages with arbitrary history size $0 \le h \le \infty$. A history size $h > 0$ means that a round k message includes also the values broadcast in the h rounds $0 \le k-h,\ldots,k-1$ prior to the current round k (if any). If some round k' message from sender q arrives at process p while in round $k \le k'$, all the still empty entries $V_p^{q,l} \in \mathcal{V}_p^l$ for $max\{k, k'-h\} \le l \le k'$ are filled with the appropriate values contained in the message. Although such perceptions are usually time faulty, in the sense that they should have arrived earlier (namely, in the process's round l message), they are nevertheless useful for some algorithms: $h = \infty$ models full information mode protocols, whereas $h = 0$ corresponds to the standard situation. The case $h > 0$ allows to model a more flexible round switching, which allows to incorporate information from processes within $h+1$ rounds. In the algorithm of Figure 1, for example, we employ $h = 2$ for *echo*-messages to substitute round l messages that were not received due to late booting or catch-up.

Formally, the essentials of the above execution pattern are captured by two specific events: $be_p^k = V_p^k[t_p^k]$ is process p's round k *broadcast event*, whereas $pe_q^{p,k} = V_q^{p,k}[t_q^{p,k}]$ denotes process q's *perception event* of process p's broadcast event. Those events are related via their parameter values $V_q^{p,k} = V_p^k$ (which are equal if there is no fault) and their occurrence times $t_q^{p,k} = t_p^k + \delta_q^{p,k}$, where $\delta_q^{p,k}$ is the *end-to-end computational + transmission delay* between sender p and receiver q in round k. Note that $\delta_q^{p,k}$ includes any round k computation at sender and receiver processes, in particular, the computation of $F_p\big(\mathcal{V}_p^k(\sigma_p^k)\big)$.

Our model stipulates lower and upper bounds $\tau^- > 0$ and $\tau^+ < \infty$, usually not known to the algorithm, which guarantee

$$\tau^- \le \delta_q^{p,k} \le \tau^+ \tag{1}$$

for any two well-behaved processes p, q connected by a non-faulty link. Note that this relation must be valid for any round k, and for $p = q$ as well. Introducing the interval $\boldsymbol{\tau} = [\tau^-, \tau^+]$, the above relation (1) can be written concisely as $\delta_q^{p,k} \in \boldsymbol{\tau}$. The resulting bound for $\delta_q^{p,k}$'s *delay uncertainty* resp. *delay ratio*, which will play a central role in our analysis, is given by $\varepsilon = \tau^+ - \tau^-$ resp. $\mathcal{P} = \tau^+/\tau^-$. Bear in mind that the algorithm of Figure 1 does not know τ^+ and τ^-, and not even ε or $\mathcal{P}$.

2.2 Perception Failure Model

Consider the round k perception vector $\mathcal{V}_q^k(t)$—observed at some real-time t— of a well-behaved process q. Our execution model implies that $\mathcal{V}_q^k(t)$ is monotonic in time, in the sense that $|\mathcal{V}_q^k(t+\Delta t)| \ge |\mathcal{V}_q^k(t)|$ for any $\Delta t \ge 0$, since perceptions are only added. Moreover, since the value V_q^{k+1} to be broadcast in the next round $k+1$ is computed solely from $\mathcal{V}_q^k := \mathcal{V}_q^k(\sigma_q^k)$ and q's local state at the round switching time σ_q^k, it is obvious that, ultimatively, only the failures in the perceptions present at the respective round switching times count. Timing failures are no longer visible here (but will probably affect σ_q^k, recall Section 2.1), since a message that did not drop in by σ_q^k at process q just results in $V_q^{p,k} = \emptyset$. Consequently, the resulting

[2] Throughout the paper, we use the following notation: "Anonymous" processes and round numbers are usually denoted by lowercase letters p, q and k, l, respectively. Process subscripts denote the process where a quantity like $V_q^{p,k}$ is locally available, process superscripts denote the remote source of a quantity. Calligraphic variables like $\mathcal{V}_p^r$ denote sets or vectors, bold variables like $\boldsymbol{\tau}$ denote intervals.

[3] Since we allowed multiple concurrent processes, there may of course be several different perception vectors on a processor. Any process may send messages for a specific perception vector, at most one process per processor may receive messages from it.

perception failure model is much simpler than the physical one and therefore more suitable for analyzing an algorithm's fault-tolerance properties.

Our formalization solely rests upon the $n \times n$ matrix $\mathcal{V}^k(t)$[4] of round k perceptions observed at the same arbitrary time t—typically, some process's round switching time—at all processes:

$$\mathcal{V}(t) = \begin{pmatrix} \mathcal{V}_1(t) \\ \mathcal{V}_2(t) \\ \vdots \\ \mathcal{V}_n(t) \end{pmatrix} = \begin{pmatrix} V_1^1 & V_1^2 & \dots & V_1^n \\ V_2^1 & V_2^2 & \dots & V_2^n \\ \vdots & \vdots & \vdots & \vdots \\ V_n^1 & V_n^2 & \dots & V_n^n \end{pmatrix}_t \tag{2}$$

Note that $\mathcal{V}(t)$ is in fact a quite flexible basis for our failure model, since different "views" of the state of the distributed computation can be produced easily by choosing a suitable t.

We distinguish the following failure modes for single perceptions in $\mathcal{V}(t)$ in our perception failure model:

Definition 1 (Perception Failures). *Process q's perception V_q^p of process p's broadcast value V_p can be classified according to the following mutually exclusive failure mode predicates:*

- $\text{correct}\big(V_q^p\big)$*:* $V_q^p = V_p$,
- $\text{omission}\big(V_q^p\big)$*:* $V_q^p = \emptyset$,
- $\text{value}\big(V_q^p\big)$*:* $V_q^p \neq \emptyset$ *and* $V_q^p \neq V_p$.

Next, we have to classify sender process failures. If $\mathcal{R}_p$ denotes some sender p's receiver set, let $\overline{\mathcal{R}}_p \subseteq \mathcal{R}_p$ denote the set of non-faulty processes among those.

Definition 2 (Perception Process Failures). *Let p be a (faulty) sender process and g be an non-faulty receiver such that $\emptyset \neq V_g^p \in \mathcal{V}(t)$. In the absence of link failures, process failures of p can be classified according to the perceptions $V_q^p \in \mathcal{V}(t+\varepsilon)$ at all non-faulty receivers $q \in \overline{\mathcal{R}}_p \subseteq \mathcal{R}_p$ as follows:*

- *Non-faulty:* $\forall q \in \overline{\mathcal{R}}_p : \text{correct}\big(V_q^p\big)$,
- *Manifest:* $\forall q, r \in \overline{\mathcal{R}}_p : V_q^p = V_r^p \neq V_p$ *detectably,*
- *Crash:* $\forall q \in \overline{\mathcal{R}}_p : \text{omission}\big(V_q^p\big)$,
- *Omission:* $\forall q \in \overline{\mathcal{R}}_p : \text{correct}\big(V_q^p\big) \vee \text{omission}\big(V_q^p\big)$,
- *Symmetric:* $\forall q, r \in \overline{\mathcal{R}}_p : V_q^p = V_r^p$,
- *Arbitrary: no constraints.*

A faulty process producing at most asymmetric omission failures is called benign faulty.

The following Definition 3 specifies the possible failures in perceptions caused by link failures.

Definition 3 (Perception Link Failures). *In the absence of sender process failures, a failure of the link from sender p to an non-faulty receiver q can be classified according to its effect upon q's perception $V_q^p \in \mathcal{V}(t)$ as follows:*

- *Link non-faulty:* $V_q^p = V_p$,
- *Link omission:* $V_q^p = \emptyset$,
- *Link arbitrary: no constraint.*

The failure classes up to link omission failures are called benign.

[4] We will subsequently suppress the round number k in quantities like $\mathcal{V}^k(t)$ for brevity.

To overcome the impossibility of consensus in presence of unrestricted link failures [7, 13], it turned out that send and receive link failures should be considered independently [25, 26]. The following link-failure-related parameters are hence incorporated in the final perception failure model of Definition 4 below:

(A1^s) *Broadcast link failures*: For any single sender s, there are at most f_ℓ^s receiver processes q with a perception vector $\mathcal{V}_q$ that contains a faulty perception V_q^s from s.

(A1^r) *Receive link failures*: In any single process q's perception vector $\mathcal{V}_q$, there are at most f_ℓ^r faulty perceptions V_q^p.

A process that suffers from at least one broadcast link failure is said to commit a *broadcast failure*, whereas a process that suffers from at least one receive link failure is said to commit a *receive failure*. Note carefully that we will allow every process in the system to commit a broadcast and/or receive failure in every round, without considering the process as faulty in the usual sense. Up to f_ℓ^s send and up to f_ℓ^r receive link failures may occur in any process's broadcast failure and receive failure, respectively, and the particular links actually hit are usually different in different rounds.

Definition 4 (Asynchronous Perception Failure Model). *For some arbitrary time t, let $\mathcal{V}^k(t)$ be the round k perception matrix of an asynchronous system of processes running on different processors that comply to our execution model. For any non-faulty receiver q, it is guaranteed that $V_q^{p,k} = \emptyset$ if $p \notin \mathcal{S}_q^k$ or if $V_q^{p,k}$ was not received by time t. Moreover:*

(P1) There are at most f_a, f_s, f_o, f_c, and f_m columns in $\mathcal{V}^k(t)$ that may contain arbitrary, symmetric, omission, crash, and manifest faulty perceptions $V_q^{p,k}$, respectively, resulting from faulty processes; up to $f_a + f_s$ may be timing faulty.

(A1^s) In each single column p, at most f_ℓ^s arbitrary perceptions $V_q^{p,k} \in \mathcal{V}^k(t)$ corresponding to non-faulty receivers $q \in \overline{\mathcal{R}}_p^k \subseteq \mathcal{R}_p^k$ may differ from the value(s) obtained in the absence of broadcast link failures. At most $f_\ell^{s\,a} \leq f_\ell^s$ of those may be link arbitrary faulty.

(A1^r) In each single row q corresponding to an non-faulty receiver, at most f_ℓ^r of the perceptions $V_q^{p,k} \in \mathcal{V}^k(t)$ corresponding to senders $p \in \mathcal{S}_q^k$ may differ from the outcome(s) obtained in the absence of receive link failures. At most $f_\ell^{r\,a} \leq f_\ell^r$ of those may be link arbitrary faulty.

(A2) Process q knows the origin p of $V_q^{p,k} \in \mathcal{V}^k(t)$.

(A3) $\emptyset \neq V_q^{p,k} \in \mathcal{V}^k(t) \Rightarrow \emptyset \neq V_r^{p,k} \in \mathcal{V}^k(t+\varepsilon)$ for every non-faulty sender p connected to non-faulty receivers q and r via non-faulty links.

2.3 Model of the Startup Phase

At the very beginning all processes are down. Every message that arrives at a process while it is down is lost, and no messages are sent by such a process, not even spurious ones generated by arbitrary link failures. Correct processes boot at unknown times that cannot be bounded a priori. For faulty processes, we assume[5] that they are up right from the beginning. Throughout the paper, we will use n_{up} for the number of processes, including the faulty ones, that are up at a given time.

During startup, a correct process goes through the following sequence of operating modes:

1. *down*: A process remains down when it has not been started yet or has not completed booting.
2. *up*: A process gets up if it has completed booting. To get a clean distinction of up and down, we assume that a process flushes the input queues of its network interface as the first action after booting is completed. Hence, the algorithm gets only messages that dropped in when it was already up.
3. *passive*: A process that just got up performs an algorithm-dependent initialization phase, where it is called passive. As the first action in passive mode, a process may send a *join* message to all its peers. The first reception of a *join* message from some process p causes the receiver to retransmit the current round message to p (point-to-point); subsequent *join* messages from the same sender are ignored.
4. *active*: A process that has completed its initialization phase is called active.

[5] We do not care about the clocks of faulty processes in this paper, since they need not satisfy the clock synchronization conditions. If we studied *uniform* variants of (P) and (A), cp. [10, 25], benign faulty ("obedient") processes must be exempted here.

In case of the algorithm of Figure 1, for example, a passive process broadcasts *join*—to get the last *(echo, k)* message of every peer—and participates in the algorithm as in active mode. It need not satisfy the clock synchronization conditions (P) and (A) while passive, however. The transition to active mode occurs when the process can be sure that it is within the synchronization precision D_{max}.

3 Algorithm and Major Results

Assuming that every process p in the system is equipped with an adjustable discrete clock $C_p(t)$ that can be read at arbitrary real-times t, a proper clock synchronization algorithm must guarantee the following:

Definition 5 (Clock Synchronization Conditions). *(P)* Precision Requirement: *There is some* precision $D_{max} > 0$ *such that*

$$|C_p(t) - C_q(t)| \leq D_{max} \tag{3}$$

for any two active non-faulty processes p and q and any real-time t.

(E) Envelope Requirement: *There are some constants $R^-, O^-, R^+, O^+ > 0$ such that*

$$O^-(t_2 - t_1) - R^- \leq C_p(t_2) - C_p(t_1) \leq O^+(t_2 - t_1) + R^+ \tag{4}$$

for any active non-faulty process p and any real-times $t_2 \geq t_1$.

The *precision requirement* (P) just states that the difference of any two correct clocks in the system must be bounded, whereas the *envelope requirement* (E) guarantees some relation of the progress of clock time with respect to the progress of real-time; (E) is also called *accuracy requirement* in the literature.

Traditional research on clock synchronization (see [16,19,27,28] for an overview) considers synchronous systems equipped with hardware clocks with high time-resolution and small drift ρ (in the range of $1 \ldots 100$ μs/s). Optimal clock synchronization algorithms for synchronous systems like [5, 18, 29] guarantee $O^- = 1 - \rho$ and $O^+ = 1 + \rho$ and very small R^-, R^+. The achievable precision depends primarily upon the transmission delay uncertainty ε, which is in the $1 \ldots 10$ ms-range for typical LANs. By contrast, clocks in partially synchronous systems [4, 15] are implemented as simple software counters. In our case, a process p's clock will be the round number of the clock synchronization algorithm running on p: $C_p(t)$ is incremented by 1 when process p switches to the next round. The time-resolution of any clock is hence determined by the number of round switches within a given real time interval. Theorem 7.5 will reveal that our algorithm ensures $O^- = 1/(2\tau^+)$, $O^+ = 1/(2\tau^-)$ with small reasonably small R^-, R^+, where the lower bound O^- and R^- holds only if sufficiently many processes are up and running.

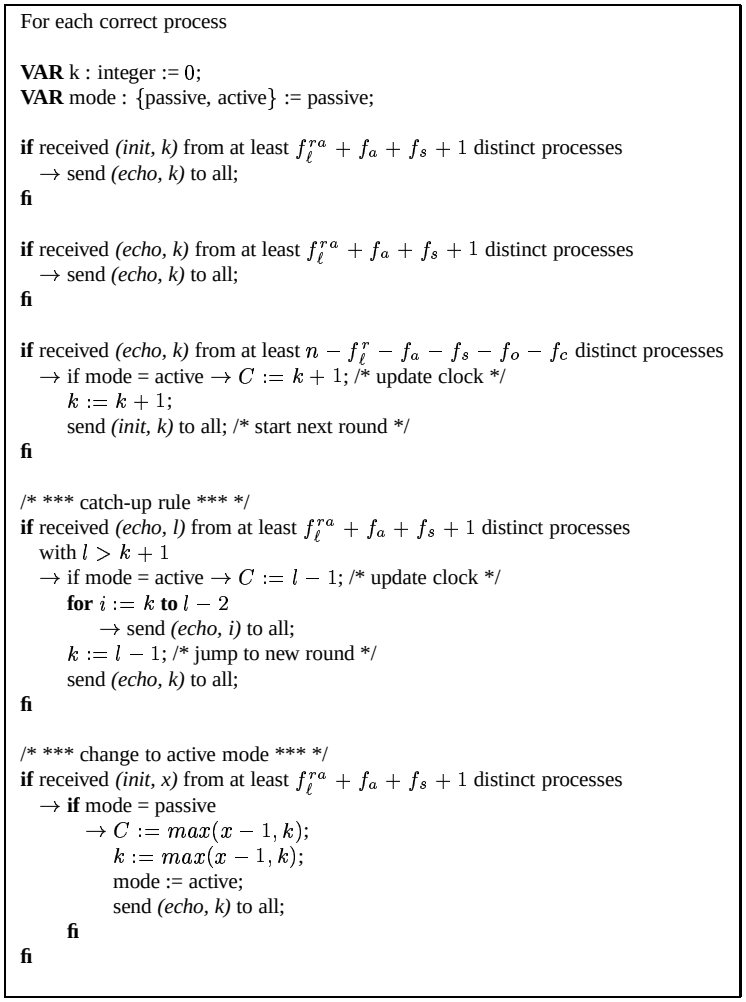
Note carefully that, by using local hardware timers in conjunction with hardware packet timestamping at all processors, it is possible to "stretch" τ^- to ensure $\tau^+ \approx \tau^- + 2\rho + \epsilon$, where ϵ is the communication delay uncertainty of the network only. This way, a situation comparable to synchronous clock synchronization settings can be established, see [21] for details.

The clock synchronization algorithm considered in this paper is a hybrid variant of the algorithm of [34]. It is an extension of the well-known non-authenticated[6] clock synchronization algorithm of [29], which employs consistent broadcasting for generating nearly-simultaneous global resynchronization events in the system. Every processor p runs two[7] concurrent processes, which maintain $C_p(t)$ by executing the pseudo-code shown in Figure 1. Comparison of our algorithm with hybrid variants [10, 21] of the original consistent broadcasting primitive (without startup handling) shows that the first three **if**-clauses are the same: Informally, each round k is started by sending an *(init, k)* message to all. If a correct process can be sure that at least one correct process has sent a round k message, it sends *(echo, k)* to all. When a process can be sure that there are enough round k messages in the system to guarantee that every correct process will eventually advance its clock, it advances to round $k + 1$ and hence sends *(init, k+1)*. This guarantees both (P) and (A) if sufficiently many correct processes are up and running in systems with $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$.

In order to properly handle system startup as well, however, *join* messages and two additional **if**-clauses are required. As shown in [34], three consecutive modes of system operation must be distinguished here:

[6] Since we consider a distributed system at the very beginning, we cannot assume any kind of authentication service.

[7] In order to comply to the formal requirements of our execution model, every processor p must run two processes P and P' that process *(init, k)* and *(echo, k)* messages in their perception vectors $\mathcal{V}_p^k$ and $\mathcal{V}_p^{k'}$, respectively. $C_p(t)$ is simply the current round number k of P', which is shared by both processes. Note that only P' needs to react to a *join* message of a newly booted processor, by retransmitting *(echo, k)*. In order to keep our description simple, we will usually not explicitly distinguish P and P', however: By saying that process p responds to *(init, k)* resp. *(echo, k)*, we mean that P resp. P' on processor p does so.



For each correct process

VAR k : integer := 0;
VAR mode : {passive, active} := passive;

if received *(init, k)* from at least $f_\ell^{ra} + f_a + f_s + 1$ distinct processes
  → send *(echo, k)* to all;
fi

if received *(echo, k)* from at least $f_\ell^{ra} + f_a + f_s + 1$ distinct processes
  → send *(echo, k)* to all;
fi

if received *(echo, k)* from at least $n - f_\ell^{r} - f_a - f_s - f_o - f_c$ distinct processes
  → if mode = active → $C := k + 1$; /* update clock */
    $k := k + 1$;
    send *(init, k)* to all; /* start next round */
fi

/* *** catch-up rule *** */
if received *(echo, l)* from at least $f_\ell^{ra} + f_a + f_s + 1$ distinct processes
  with $l > k + 1$
  → if mode = active → $C := l - 1$; /* update clock */
    for $i := k$ **to** $l - 2$
      → send *(echo, i)* to all;
    $k := l - 1$; /* jump to new round */
    send *(echo, k)* to all;
fi

/* *** change to active mode *** */
if received *(init, x)* from at least $f_\ell^{ra} + f_a + f_s + 1$ distinct processes
  → **if** mode = passive
      → $C := max(x - 1, k)$;
        $k := max(x - 1, k)$;
        mode := active;
        send *(echo, k)* to all;
    fi
fi

Figure 1. Clock Synchronization Algorithm for the Hybrid Perception-based Failure Model with Startup Phase

- *Early mode*, where the first few correct processes have completed booting and started exchanging messages.
- *Degraded mode*, where enough correct processes are up such that some clocks may advance when "assisted" by faulty processes or links.
- *Normal mode*, where sufficiently many correct processes are up and synchronized to guarantee progress for all clocks.

Note carefully that it is impossible for any process in the system to delimit the exact borders between those modes from local information.

First of all, a newly booted process must tell all others that it up now and must learn their current clock values. This is accomplished by means of *join* messages, as introduced in Section 2.3: Every process p sends *join = (echo, 0)* as the very first message after having completed booting. Every process q that receives this message replies by retransmitting its previously sent *(echo, k)* message. This ensures that p will eventually get sufficiently many messages—which may have been lost while it was down—to trigger the catch-up rule described below.

The major problem in degraded mode is the impossibility to guarantee (P) solely via the third **if**-clause: There are not sufficiently many correct processes to guarantee that every correct process will eventually advance its clock when a single one does so. Here it is where the fourth **if**, our *catch-up rule*, comes into play: It allows a correct process to advance its clock to round $k - 1$ when sufficiently many round k messages have been received. Therefore, eventually, a sufficiently large group

of correct processes can be guaranteed to be within two rounds of each other. This causes two other problems, however: First, the second and third **if**-clause must trigger when sufficiently many *echo*-messages from different processes within 3 rounds have been received. This is conveniently expressed in our execution model by setting the history size of *(echo, k)* messages to $h = 2$, recall Section 2.1: Since the reception of *(echo, k)* at process q implies that the sender p must already have sent *(echo, l)* for all $l < k$ as well, empty round $k-1$ and $k-2$ perceptions $V_q^{p\prime}$ can safely be filled on that occasion as well. Note that those perceptions may be empty if q missed the earlier messages due to late booting.

Second, due to failures or a large group of simultaneous late joiners, the first round an initializing process may reach by the catch-up rule could be arbitrarily small. This would violate the precision requirement, however. We therefore assume that only active, but not passive processes, must satisfy (P) and (A). A passive process switches to active mode when it has sufficient evidence that its local round number is within D_{max} of the clocks of the other correct active processes in the system. This is accomplished by the fifth **if**-clause, which triggers when sufficiently many *(init, x)*-messages for an arbitrary round x have been obtained. Since it can be shown that x must be sufficiently close to the maximum correct clock value in the system since *init*-messages are only generated by the third **if**-clause, this conditions indeed serve their purpose.

Results: Due to space restriction we briefly summarize the major results. See the appendix for a detailed analysis.

As far as precision (P) is concerned, our algorithm always guarantees a precision (Theorem 5.5) of $D_{max} = \lfloor 2\mathcal{P} + 5/2 \rfloor$ for all correct active processes. This worst case precision, however, applies only to some processes during a short period of time, namely, late starters during the first ε seconds of active mode. The clock value of such a process becomes better as soon as the information from all other processes arrives. Since there are two rules that can be used by a process to advance its clock (third and fourth **if**-clause in Figure 1), there are two bounds on the achievable precision after this worst case situation. The first of those is enforced by the fourth **if** (Theorem 4.6): As soon as all answers to *join* message reach an initializing process, it becomes synchronized to within $D_{MCB} = \lfloor 1/2\mathcal{P} + 5/2 \rfloor$ even if not all correct processes are up at that time, i.e., even in degraded mode. During normal mode, when all correct processes are active, the third **if** is guaranteed to be enabled within bounded time at every correct process (see below). It enforces another bound on the precision (Theorem 7.6) $D'_{MCB} = \lfloor 3/2\mathcal{P} + 1/2 \rfloor$

The time required to enter normal operation after sufficiently many correct processes got up is bounded (Theorem 5.7) by $\Delta_{init} = 8\tau^+$. A precision of $\min\{D_{MCB}, D'_{MCB}\}$ can be guaranteed in normal mode after that time. Which of the contributing bounds is tighter depends solely on the network behaviour, i.e. on the ratio of τ^+ and τ^-. In normal mode our algorithm also guarantees (A) accuracy (Theorem 7.5) $\frac{t_2-t_1}{2\tau^+} - 4 + \frac{1}{\mathcal{P}} < C_p(t_2) - C_p(t_1) < \frac{t_2-t_1}{2\tau^-} + D_{max} + 1$ for all active processes. The upper envelope bound is guaranteed to hold during system startup as well, i.e., clocks cannot progress faster during the whole system life time. The lower envelope bound, however, could be 0 during system startup, i.e., there need not be progress of the clocks in degraded mode.

4 From Early to Degraded Mode

Before considering the full system startup scenario, we will study the algorithm of Figure 1 in a system of $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ processes with a fixed but arbitrary number $0 \leq n_{up} \leq n$ of running processes. We will assume that all n_{up} processes, except the at most $f_a + f_s$ arbitrary or symmetrically faulty ones, are initially synchronized and active. Our results will hence show how the algorithm works with a small number of participating processes if we assume initial synchronization. It will turn out in Section 5, however, that the above minimum number of processes in the system must be increased by $f_s + f_c$ when full system startup is considered.

We start with a few definitions and lemmas, which will be used frequently in our analysis. First of all, the following properties of the perception vectors at two different non-faulty receivers follow immediately from our fault model in Definition 4.

Lemma 4.1 (Difference in Perceptions [22]). *At any time t, the perception vector $\mathcal{V}_q(t)$ of any process on an non-faulty receiver q may contain at most $f_\ell^{ra}+f_a+f_s$ time/value-faulty perceptions $V_q^p \neq \emptyset$. Moreover, at most $\Delta f = f_\ell^{ra}+f_\ell^r+f_a+f_o$ perceptions V_r^p corresponding to $V_q^p \neq \emptyset$ may be missing in any other non-faulty receiver's $\mathcal{V}_r(t+\Delta t)$ for any $\Delta t \geq \varepsilon$.*

Proof. The first statement of our lemma is an obvious consequence of Definition 4. To prove the second one, we note that at most $f_\ell^{ra} + f_a$ perceptions may have been available (partly too early) at q without being available yet at r, additional $f_\ell^{ra\prime} \leq f_\ell^{ra}$ perceptions may be late at r, and $f_\ell^r - f_\ell^{ra\prime}$ ones could suffer from an omission at r. Summing up all the differences, the expression for Δf given in Lemma 4.1 follows. □ □

Our analysis will heavily use properties related to the maximum clock value of all well-behaved processes in the system, which is given by the following Definition 6.

Definition 6 (Maximum Local Clock Value). *Let $C_{max}(t)$ denote the maximum of all local clock values of correct or benign faulty processes that are up at time t. Let further $\sigma^k_{first} = \sigma^k_p \leq t$ be the real-time the first correct or benign faulty process p has set its local clock value to $k+1 = C_{max}(t)$.*

The following Lemma 4.2 shows that progress of $C_{max}(t)$ is only possible via the third **if** in Figure 1. This fact will be heavily used in our proofs.

Lemma 4.2 (3rd if). *Every correct process that sets its clock to $C_{max}(t)$ by time t must do so by the third* **if** *clause.*

Proof. By contradiction. Let a correct process p set its clock to $k = C_{max}(t)$ at instant t by a catch-up rule (fourth or fifth **if** clause). At least one correct (or benign faulty) process must have sent a message for a tick $l > k$ to enable the catch-up rule at p. Since such processes only send messages for ticks less or equal their local clock value at least one must have had a clock value $l > k$ at instant t. Thus $C_{max}(t) > k$ which provides the required contradiction. □ □

The following Theorem 4.3 is the first major result of this section. It shows that the clocks at correct processes obey three properties, namely, weak correctness (P1W), unforgeability (P2), and weak relay (P3W), the names of which have been coined in the context of consistent broadcasting [10, 29, 30]. Our properties are weak ones, however, since we do not have sufficiently many processes up and running to ensure the strong ones of Theorem 7.1. Note that our theorem is even valid for $n_{up} < n - f_a + f_s + f_o + f_c$, although it is trivial to see from Figure 1 that no correct process can make any progress in this case.

Theorem 4.3 (Weak Clock Synchronization Properties). *For $n \geq 2f^{ra}_\ell + 2f^r_\ell + 3f_a + 2f_s + 2f_o + f_c + 1$ and any n_{up}, the algorithm from Figure 1 achieves*

P1W Weak Correctness. If at least $f^{ra}_\ell + f^r_\ell + f_a + f_s + 1$ correct processes set their clocks to k by time t, then every correct process sets its clock to $k-1$ by time $t + 2\tau^+$.

P2 Unforgeability. If no correct or benign faulty process sets its clock to k by time t, then no correct process sets its clock to $k+1$ by $t + 2\tau^-$ or earlier.

P3W Weak Relay. If a correct process sets its clock to k at time t, then every correct process sets its clock to $k-2$ by time $t + \varepsilon$.

Proof. Weak Correctness. If $k = C_{max}(t)$, at least $f^{ra}_\ell + f^r_\ell + f_a + f_s + 1$ correct processes must have sent *(init, k)* by time t according to Lemma 4.2. The perception vectors of these $f^{ra}_\ell + f^r_\ell + f_a + f_s + 1$ correct processes hence satisfy $|\mathcal{V}^k(t+\tau^+)| \geq f^{ra}_\ell + f_a + f_s + 1$. Hence, they all achieve sufficient evidence in the first **if** by time $t + \tau^+$, where they send *(echo,k)* to all processes. By time $t + 2\tau^+$ all correct processes reach sufficient evidence in the fourth **if** to catch-up.
If $k < C_{max}(t)$, at least $f^{ra}_\ell + f^r_\ell + f_a + f_s + 1$ correct processes must have set their clocks to k before some time $t'' < t$ using the third **if** according to Lemma 5.1. Consequently, all correct processes set their clocks to $k-1$ by time $t'' + 2\tau^+ < t + 2\tau^+$ by the same reasoning as above.

Unforgeability. Setting the clock value can be done at a process by (1) the third **if**, by (2) the fourth **if** or (3) by the fifth **if**. The proof for both is by contradiction.
Assume (1) that there is a process p that sets its clock to $k+1$ before instant $t + 2\tau^-$ using the third **if**. This implies $|\mathcal{V}^{k'}_p(t + 2\tau^-)| \geq 2f^{ra}_\ell + f^r_\ell + 2f_a + f_o + f_s + 1$.
Since only at most $f^{ra}_\ell + f_a + f_s$ of the corresponding *(echo, k)* messages may be due to messages produced by arbitrary receive link faults and time/value–faulty processes at least one correct process q must have sent[8] such an *(echo, k)* message before time $t + \tau^-$. Process q has done so caused by received messages for the k^{th} tick such that $|\mathcal{V}^k_q(t+\tau^-)| \geq f^{ra}_\ell + f_a + f_s + 1$ or $|\mathcal{V}^{k'}_q(t+\tau^-)| \geq f^{ra}_\ell + f_a + f_s + 1$. Hence—by the same argument as before—at least one correct process must have sent a message for the k^{th} tick before time t, which contradicts the assumption of unforgeability.
Assume (2) that there is a process p that sets its clock to $k+1$ before instant $t + 2\tau^-$ using the fourth **if**. p does so because $|\mathcal{V}^{l'}_p(t + 2\tau^-)| \geq f^{ra}_\ell + f_a + f_s + 1$ or for some $l > k+1$. That is, at least one *(echo, l)* message must have been sent by a correct process q before $t + \tau^-$. Process q must have sent it, because $|\mathcal{V}^x_q(t+\tau^-)| \geq f^{ra}_\ell + f_a + f_s + 1$ or $|\mathcal{V}^{x'}_q(t+\tau^-)| \geq f^{ra}_\ell + f_a + f_s + 1$ for a tick $x \geq l$. These perceptions stem from messages that must have been sent before

[8] Note that this statement remains true if q sent *(echo, $k+h$)* and hence generates a "simulated" reception of *(echo,k)*, according to the semantics of messages with history, recall Section 2.3.

time t since $x > k$ and messages for tick x are sent by a correct process only after tick k messages. But by P2's assumption no tick k message was sent before t, which again provides the required contradiction.
Assume (3) that there is a process p that sets its clock to $k+1$ before instant $t + 2\tau^-$ using the fifth **if**. Process p does so because $|\mathcal{V}_p^{k+2}(t+2\tau^-)| \geq f_\ell^{ra} + f_a + f_s + 1$. At least one *(init, $k+2$)* message must have been sent by a correct process q before $t + \tau^-$. Process q must have sent it, because $|\mathcal{V}_q^{(k+1)'}(t+\tau^-)| \geq 2f_\ell^{ra} + f_\ell^r + 2f_a + f_o + f_s + 1$, such that more than one correct processes must have sent *(echo, $k+1$)* before time t. A correct process never sends any $k+1$ messages before it has send a message for tick k. By assumption no tick k message was sent by time t which again provides the contradiction.

Weak Relay. Assume $k = C_{max}(t)$. The correct process p that advanced its clock to k by assumption must use the third **if** according to Lemma 4.2, such that $|\mathcal{V}_p^{(k-1)'}(t)| \geq 2f_\ell^{ra} + f_\ell^r + 2f_a + f_o + f_s + 1$. Hence the perception vector for tick $k-1$ at any correct process q at time $t + \varepsilon$ must satisfy $|\mathcal{V}_q^{(k-1)'}(t+\varepsilon)| \geq f_\ell^{ra} + f_a + f_s + 1$ according to Lemma 4.1. It follows that all correct processes achieve sufficient evidence in the fourth *if* to catch-up to round $k-2$.
If $k < C_{max}(t)$ then at least one correct process has already set its clock to $l > k$ at $t' \leq t$ using the third *if*. We have shown in the previous paragraph that all correct processes must set their clocks to $l-2$ by time $t' + \varepsilon$ so all correct processes must have set their clocks to $k-2$ by time $t + \varepsilon$ are well. □ □

The following simple Lemma 4.4, which will be used frequently in our proofs, follows immediately from property P2. Note that it is hence true for any clock synchronization algorithm that satisfies unforgeability.

Lemma 4.4 (Fastest Progress). *Let p be the first correct process that sets its clock to k at time t. Then no correct process can can reach a larger clock value k' before $t + 2\tau^-(k'-k)$.*

Proof. By induction on $l = k' - k$. For $l = 1$ Lemma 4.4 is identical to unforgeability and therefore true. Assume that no correct process has set its clock to $k+l$ before $t + 2\tau^- l$ for some l. Thus no processes may set their clocks to $k+l+1$ before $t + 2\tau^- l + 2\tau^- = t + 2\tau^-(l+1)$ following unforgeability. Hence Lemma 4.4 is true for $l+1$ as well. □ □

In our analysis we will frequently require to bound the increase of C_{max} during a given real-time interval $[t_1, t_2]$. For this purpose Lemma 4.4 can only be applied if and only if $t_1 = \sigma_{first}^k$. In all other cases we cannot use Lemma 4.4. As in [23] we unify this situation, starting with the following Definition 7.

Definition 7 (Synchrony). *Real-time t is in synchrony with $C_{max}(t)$ iff $t = \sigma_{first}^k$ for some real-time σ_{first}^k (for some arbitrary k) as defined in Definition 6. Let the indicator function of non-synchrony be defined as*

$$I_{t \neq \sigma} = I_\sigma(t) = \begin{cases} 0 \text{ if } t \text{ is in synchrony with } C_{max}(t), \\ 1 \text{ otherwise}. \end{cases}$$

Lemma 4.5 (Maximum Increase of C_{max} within Time Interval). *Given any two real-times $t_2 \geq t_1$. $C_{max}(t_2) - C_{max}(t_1) \leq \lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor + I_\sigma(t_1)$.*

Proof. Let $k = C_{max}(t_1) - 1$. We have to distinguish the two cases $\sigma_{first}^k = t_1$ and $\sigma_{first}^k < t_1$.

Let $\sigma_{first}^k = t_1$ such that $I_\sigma(t_1) = 0$. From Lemma 4.4 follows that C_{max} may increase every $2\tau^-$ time-units hence $\lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor$ times before t_2. Since $I_\sigma(t_1) = 0$ Lemma 4.5 is true for this case.

Now let $\sigma_{first}^k < t_1$ such that $I_\sigma(t_1) = 1$. And let the real-time $t' = \sigma_{first}^{k+1} > t_1$. We can now apply Lemma 4.4 starting from time t'. Since $t_2 - t_1 > t_2 - t'$ it follows from Lemma 4.4 that C_{max} cannot increase more often than $\lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor$ times between t' and t_2. At instant t' C_{max} has already increased once such that $C_{max}(t_2) - C_{max}(t_1) \leq \lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor + 1$. Since $I_\sigma(t_1) = 1$ Lemma 4.5 is also true for this case. □ □

In the following major Theorem 4.6 we give a bound for the precision requirement (P). We assume an instant t such that for a correct process p $\sigma_p^k = t$. From weak relay we can conclude about a bound for σ_q^{k+2} for any other correct process q such that $\sigma_q^{k+2} = \sigma_{first}^{k+2}$. Using Lemma 4.5 we can give a bound for $C_{max}(t)$ and hence precision.

Theorem 4.6 (Precision in Degraded Mode). *In a system with $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ processes, where $0 \leq n_{up} \leq n$ processes, except the faulty ones, are initially synchronized, the algorithm of Figure 1 satisfies the precision requirement (P) with $D_{MCB} = \lfloor \frac{1}{2}\mathcal{P} + \frac{5}{2} \rfloor$.*

Proof. If no correct process advances its clock beyond $k = 2$, precision $D_{MCB} \geq 2$ is automatically maintained since all clocks are initially synchronized to $k = 0$.

Assume a correct process p has a local clock value $k \geq 0$ within an unknown precision D_{MCB} to all other correct processes—and therefore also to $C_{max}(t')$—at real-time t'. We now use weak relay (P3W), Definition 7 and Lemma 4.5 to reason about D_{MCB} by calculating $C_{max}(t)$ for a time $t > t'$.

Let p advance its clock to $k+1$ such that $\sigma_p^k = t > t'$. Process p has not done so before t because no other correct process has set its clock to $k+3$ before $t - \varepsilon$ following directly from weak relay (P3W), thus $\sigma_{first}^{k+2} \geq t - \varepsilon$.

From Lemma 4.5 follows that $C_{max}(t) \leq \lfloor \frac{t-(t-\varepsilon)}{2\tau^-} \rfloor + I_\sigma(t-\varepsilon) + C_{max}(t-\varepsilon)$. Let us now take a closer look at the term $I_\sigma(t-\varepsilon) + C_{max}(t-\varepsilon)$: If $\sigma_{first}^{k+2} = t - \varepsilon$ and therefore $t - \varepsilon$ is synchronized with C_{max}, $C_{max}(t-\varepsilon) = k+3$ and $I_\sigma(t-\varepsilon) = 0$ (following Definition 7). If on the other hand $\sigma_{first}^{k+2} > t - \varepsilon$, $C_{max}(t-\varepsilon) = k+2$ and $I_\sigma(t-\varepsilon) = 1$. In both cases $I_\sigma(t-\varepsilon) + C_{max}(t-\varepsilon) = k+3$ such that $C_{max}(t) \leq \lfloor \frac{\varepsilon}{2\tau^-} \rfloor + k + 3$ thus $C_{max}(t) \leq \lfloor \frac{1}{2}\mathcal{P} + \frac{5}{2} \rfloor + k$.

Process p has a clock value $C_p(t') = k$ at a time $t' < t$ which is by assumption within precision. Since $C_p(t') < C_p(t)$ and $C_{max}(t') \leq C_{max}(t)$, we get a bound for our precision D_{MCB} from the difference $C_{max}(t) - C_p(t') = C_{max}(t) - k \leq \lfloor \frac{1}{2}\mathcal{P} + \frac{5}{2} \rfloor$. □ □

Remark Note carefully that, although Theorem 4.6 is written in the context of the algorithm of Figure 1, it actually holds for every algorithm that satisfies weak correctness (P1W), unforgeability (P2), and weak relay (P3W).

The result of Theorem 4.6 can be applied to all correct processes that participate in the algorithm during the early phase, i.e., boot early. All those processes remain synchronized to each other within D_{MCB}, even in the subsequent degraded and normal mode. The problem is, however, that other correct processes might complete booting during degraded mode, after some progress has been achieved by the group of early starters. As a late process starts with a clock value of 0, it is clearly not synchronized. We will deal with this situation in the following section.

5 From Degraded to Normal Mode

In this section, we will incorporate correct processes that get up late, namely, during degraded mode. It will turn out that those late starters considerably spoil the overall precision in the system. This is due to the fact that such processes compute their first clock values from very little information on the system state: It could be that a late starter becomes active due to an *(init, x)* message sent by a single well-behaved process only ("assisted" by faulty processes). Although this is sufficient to guarantee some precision D_{max}, it is nevertheless true that D_{max} is considerably larger than D_{MCB}. Fortunately, even a late starter will get synchronized within precision D_{MCB} as soon as the messages from all correct processes arrive. Hence, D_{max} applies only to late starters during a short period after becoming active.

Besides the large precision D_{max}, initialization introduces another disadvantageous property: In order to handle system booting, our algorithm requires an increased number of processes for some restricted failure modes. To guarantee that every correct process eventually becomes active, sufficiently many *(init, x)* messages must be sent for every round. Lemma 5.1 will show that this requires $f_s + f_c$ additional processors such that, from now on, we have to consider a system with $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$. Symmetric process failures are hence as severe as asymmetric ones, and crash failures are as severe as our crash failures here. Note that this can be explained by the fact that processes boot at unpredictable times, which turns otherwise consistently perceived process failures in inconsistent ones.

This section also provides the analysis of the system behaviour starting from the time when the last correct process p got up. We will see that p forces all correct processes to advance their clocks, such that sufficiently many correct processes send an *(init, x)* message. As a consequence, all remaining correct processes will become active within some bounded initialization time Δ_{init}. From then on it is guaranteed that all correct processes regularly increase their clocks; system startup is completed and the system is in normal operation.

This section's results—precision D_{max} and initialization time Δ_{init}—strongly depend on the properties of *(init, k)* messages, in particular, their arrival times. The following Lemma 5.1 shows that sufficiently many of those messages are sent for every tick, such that every correct process could eventually switch to active by the fifth **if**.

Lemma 5.1 (Minimal Number of Init Messages). *Given an arbitrary point in time t with $l = C_{max}(t)$, let $t' > t$ be the instant when C_{max} further increases. For $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$, at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes set their clocks to $C_{max}(t)$ by the third* **if** *and therefore send (init, l) before t'.*

Proof. In order for the first correct process to set its clocks to $l+1$, it must have got $n - f_\ell^r - f_a - f_s - f_o - f_c \geq 2f_\ell^{ra} + f_\ell^r + 2f_a + 2f_s + f_o + f_c + 1$ *(echo, l)* messages. At least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ of those must originate from correct processes, which must have set their local clock to l by the third **if** and sent *(init, l)* earlier, recall Lemma 4.2. □ □

Another important property—expressed in the following Lemma 5.2—of our solution of initialization is the real-time t_{sync} when a late starter process p arrives at a local clock value $C_p(t_{sync})$ within precision D_{MCB} such that $C_{max}(t) - C_p(t) \leq D_{MCB}$ for any time $t \geq_{sync}$. Note carefully that we do not give an upper bound on t_{sync} w.r.t t_{up}. By now[9] it is only interesting how large C_{max} is compared to the the time p started listening. Note that the instant $t_{up} - \tau^-$ is more important than t_{up} for these properties since every message sent by a correct process after $t_{up} - \tau^-$ is guaranteed to reach p (except those messages that suffer from at most f_ℓ^r link omissions). We will use this lemma in the proofs that guarantee precision. There we must assume that processes send messages based on the fewest information. Hence they are assumed to send these messages shortly after booting possibly based on only one message by a correct processes. But they must send their worst messages before t_{sync}, i.e. before they catch up.

Lemma 5.2 (First Synchronization). *Let p be a correct processes that becomes up at instant t_{up}. And let $k = C_{max}(t_{up} - \tau^-) + 1$. Further let C_{max} increase at time t' such that $t' = \sigma_{first}^k$. From time $t_{sync} = t' + \varepsilon$ on p's local clock value $C_p(t) \geq C_{max}(t) - D_{MCB}$ for all times $t \geq t_{sync}$.*
$C_{max}(t_{sync}) \leq C_{max}(t_{up} - \tau^-) + \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor$.

Proof. Let the first correct process q set its clock to $k+1 = C_{max}(t_{up} - \tau^-) + 2$ at time $t' = \sigma_q^k = \sigma_{first}^k$. It has done so because $|\mathcal{V}_q^{k'}(t')| \geq n - f_\ell^r - f_a - f_s - f_o - f_s \geq 2f_\ell^{ra} + f_\ell^r + 2f_a + 2f_s + f_o + f_c + 1$. Thus at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct process have sent *(echo, k)* after $t_{up} - \tau^-$ because the first correct processes has set its clock to k after that and correct processes never send messages for greater ticks than their own clock value. Therefore p receices all messages sent by correct processes (except those that suffer from at most f_ℓ^r link omissions) for all ticks $l \geq k$, such that (P1W), (P2) and (P3W) can be applied to p starting from tick $k+1$. Therefore—caused by weak relay (P3W)— process p catches up by time $t_{sync} = t' + \varepsilon$, and $C_p(t) \geq C_{max}(t) - D_{MCB}$ for all times $t \geq t_{sync}$. Hence the first part of Lemma 5.2 is true.

By assumption $C_{max}(t') = C_{max}(t_{up} - \tau^-) + 2 = k + 1$ and $t' = \sigma_{first}^k$ such that following Definition 7 $I_\sigma(t') = 0$. From Lemma 4.5 follows that $C_{max}(t' + \varepsilon) \leq C_{max}(t') + \lfloor \frac{t' - (t' - \varepsilon)}{2\tau^-} \rfloor + I_\sigma(t') = C_{max}(t_{up} - \tau^-) + 2 + \lfloor \frac{1}{2}\mathcal{P} - \frac{1}{2} \rfloor = C_{max}(t_{up} - \tau^-) + \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor$. Since $t_{sync} \leq t' + \varepsilon$ Lemma 5.2 is true □ □

We now consider the quality of *(init, k)* messages when sent by correct (or benign faulty) processes. The difference of the clock value sent via an *(init, x)* message and C_{max} is in fact the most important part for calculating the precision of the clocks. We start with Lemma 5.3, which tells us that messages from the same process are required to advance C_{max} and to send the worst possible *(init, x)* message at different processes.

Lemma 5.3. *For $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$, any two correct or benign faulty processes p and q that send (init, x) and (init, y), respectively, must have got at least one message from a common process r that is either correct or benign faulty (omission or crash fault).*

Proof. Both processes p and q must have received at least $n - f_\ell^r - f_a - f_s - f_o - f_c$ *(echo)* messages in order to send *(init)*. Among those could be $f_a + f_s + f_\ell^{ra}$ messages with malign faulty origin such that $m = n - f_\ell^r - f_\ell^{ra} - 2f_a - 2f_s - f_o - f_c$ is the minimal number of messages sent by processes that are either correct or benign faulty (crash or omission). The total number of correct or benign faulty processes in the system is $n - f_a - f_s$. Since

$$2m - n + f_a + f_s = n - 2f_\ell^r - 2f_\ell^{ra} - 3f_a - 3f_s - 2f_o - 2f_c \geq 1$$

the pigeonhole principle reveals that at least one of p and q's *(echo)* messages must originate from the same correct or benign faulty process r (although it need not be the same message from r at p and q). □ □

We are now ready to to bound the quality of any *(init, k)* message from a correct process with respect to C_{max}. The worst possible *(init, k)* message is sent by process p that becomes up shortly before it sends *(init, k)*, such that is has very few information on the system state. On the other hand it must send shortly before it is guaranteed to get a good picture. These two properties are already discussed in Lemma 5.2. On the other hand we have to bound the quality of the messages—when sent by correct processes—which are used by p to calculate its clock value. In the following Lemma 5.4 we use Lemma 5.3 for this purpose.

[9] The bound on initialization time is given by Theorem 5.7

Lemma 5.4. *Every correct or benign faulty process p that sends (init, k) at instant t does so for $k \geq C_{max}(t) - \lfloor \mathcal{P} - \frac{5}{2} \rfloor$.*

Proof. We have to assume that p sends *(init, k)* on possibly few and bad information on the system state. Still Lemma 5.3 ensures that at least one message p uses is sent by another correct (or benign faulty) process r whose messages are also used ot increase C_{max}.

Let t_{up} be the time when process p changes to up. Process p sends the worst possible *(init, k)* message at instant $t \geq t_{up}$ based on an *(echo, k+1)* message sent by a correct or benign faulty process r that is received at p at time $t_1 \geq t_{up}$. We assume r to be the common process whose message was used to increase C_{max} at instant $t_2 \geq t_1 - \varepsilon$ such that $t_2 = \sigma_{first}^{k+1}$.

The worst case is that r has sent this messages as far as possible in the past, such that C_{max} could have increased as much as possible before p became up (in fact the interresting instant is $t_{up} - \tau^-$ since messages sent after this time must be received by p, faults excluded). Thus we must assume: $t_1 = t_{up}$ and $t_2 = t_{up} - \varepsilon$. Since $t_2 = \sigma_{first}^{k+1}$, t_2 is synchronized with C_{max} and hence $I_\sigma(t_2) = 0$ following Definition 7 it follows from Lemma 4.5 that $C_{max}(t_{up} - \tau^-) \leq C_{max}(t_2) + \lfloor \frac{(t_{up} - \tau^-) - (t_2)}{2\tau^-} \rfloor + I_\sigma(t_2) = k + 2 + \lfloor \frac{\varepsilon - \tau^-}{2\tau^-} \rfloor = \lfloor \frac{1}{2}\mathcal{P} + 1 \rfloor + k$.

Now we have a bound for $C_{max}(t_{up} - \tau^-)$. We must further assume that p sends *(init, k)* as late as possible before it must catch up and C_{max} increases meanwhile. Lemma 5.2 provides us a bound for $C_{max}(t_{sync})$—t_{sync} being the time p catches up—w.r.t $C_{max}(t_{up} - \tau^-)$. We must just consider that p sends *(init, k)*—based on messages with malign origin or sent by passive processes—shortly before t_{sync} in order to proof our lemma. From Lemma 5.2 follows that $C_{max}(t_{sync}) \leq C_{max}(t_{up} - \tau^-) + \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor \leq \lfloor \frac{1}{2}\mathcal{P} + 1 \rfloor + k + \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor = \lfloor \mathcal{P} + \frac{5}{2} \rfloor + k$ such that $k \geq C_{max}(t_{sync}) - \lfloor \mathcal{P} - \frac{5}{2} \rfloor$ as asserted. □ □

Equipped with Lemma 5.4, we can now bound the resulting overall precision. This precision stems from the fact that a late starter process p may change to active based on only one *(init, k)* message sent by a correct processes. Again we must assume that this happens shortly after booting, such that p's information on the system stems only from this *(init, k)* message. For the worst case we assume that a correct process sends *(init, k)* possibly far away in the past, such that C_{max} can increase as much as possible before a late started process p becomes up and receives the message. Then again Lemma 5.2 provides us with a bound on C_{max} at the instant p changes to active, shortly before it catches up.

Theorem 5.5 (Precision). *For $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$, the algorithm of Figure 1 achieves the precision property (P) during the whole system life-time with $D_{max} = \lfloor 2\mathcal{P} + \frac{11}{2} \rfloor$.*

Proof. We have to assume that p changes to active on possibly few and bad information on the system state. Still Lemma 5.4 provides us with the worst possible *(init, k)* message correct processes may send w.r.t. C_{max}.

Let t_{up} be the time when process p becomes up. Process p changes to active based on an *(init, k)* message sent by a correct or benign faulty process r that is received at p at time $t_1 \geq t_{up}$.

The worst case is that r has sent this messages as far as possible in the past at instant t_2, such that C_{max} could have increased as much as possible before p became up (in fact the interresting instant is $t_{up} - \tau^-$ since messages sent after this time must be received by p, faults excluded). Thus we must assume $t_1 = t_{up}$ and $t_2 = t_{up} - \tau^+$. From Lemma 4.5 follows that $C_{max}(t_{up} - \tau^-) \leq C_{max}(t_2) + \lfloor \frac{(t_{up} - \tau^-) - (t_2)}{2\tau^-} \rfloor + I_\sigma(t_2)$. Since t_2 is not syhnchronized with C_{max} $I_\sigma(t_2) = 1$ following Definition 7, a bound for $C_{max}(t_2)$ w.r.t. k is given by Lemma 5.4 which states that $C_{max}(t) \leq k + \lfloor \mathcal{P} + \frac{5}{2} \rfloor$. Thus $C_{max}(t_{up} - \tau^-) \leq k + \lfloor \mathcal{P} + \frac{5}{2} \rfloor + \lfloor \frac{\varepsilon}{2\tau^-} \rfloor + 1 = k + \lfloor \frac{3}{2}\mathcal{P} + 3 \rfloor$.

Now we have a bound for $C_{max}(t_{up} - \tau^-)$. We must further assume that p changes to up as late as possible before it must catch up. Lemma 5.2 provides us a bound for $C_{max}(t_{sync})$—t_{sync} being the time p catches up—w.r.t $C_{max}(t_{up} - \tau^-)$. We must just consider that p changes to active—based on messages with malign origin—at time t_{act} shortly before t_{sync} in order to proof our lemma. From Lemma 5.2 follows that $C_{max}(t_{sync}) \leq C_{max}(t_{up} - \tau^-) + \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor \leq k + \lfloor \frac{3}{2}\mathcal{P} + 3 \rfloor + \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor = \lfloor 2\mathcal{P} + \frac{9}{2} \rfloor + k$ such that $k \geq C_{max}(t_{sync}) - \lfloor 2\mathcal{P} + \frac{9}{2} \rfloor$. Since $k - 1 = C_p(t_{act})$, the value of D_{max} given in our theorem follows. □ □

We now start the analysis of the second problem of initialization: Is it guaranteed to occur within bounded time? We will show that the time Δ_{init} it takes for all correct processes to become active after the last correct one got up is bounded. For calculating Δ_{init} it is required to know how good the local clock value of the last correct process is after all answers reach it: The catch-up rule requires $f_\ell^{ra} + f_a + f_s + 1$ messages. We will see in the following Lemma 5.6 that at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes are always have clock values of $C_{max}(t)$ or $C_{max}(t) - 1$ and that therefore the clock value of the initializing process is very good after answers by these processes in front arrive.

Lemma 5.6 (Frontier Processes). *In any execution of the algorithm of Figure 1, there are at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes with local clock values $C_{max}(t)$ or $C_{max}(t) - 1$ at any time t.*

Proof. Let p be any correct process that sets its clock to the maximum at time t, such that $C_p(t) = C_{max}(t) = k$. According to Lemma 4.2, this happens via the third **if**, so p has received at least $n - f_\ell^r - f_a - f_s - f_o - f_c \geq 2f_\ell^{ra} + f_\ell^r + 2f_a + 2f_s + f_o + f_c + 1$ *(echo, $k-1$)* messages, i.e. at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ ones sent by correct processes. Correct processes never send *(echo)* messages for ticks larger than their local clock values. Therefore, at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes must have a clock value of $C_p(t)$ or $C_p(t) - 1$ at time t. □ □

We now give a bound for the maximum time interval it takes to get progress into the system after the last correct processes has become running. The worst case scenario here is different from that of the worst difference in clock values (Theorem 5.5). Whereas, for the biggest difference in clock values, the initialization must happen very quickly, so that the initializing process has very few time to gain information on the system state, we now must consider that the processes are not making progress. We must further assume that their local clock values are spread over several rounds. In this case the messages by the initializing process are required to let all correct processes update their clocks first. When they do so, the necessary *(init)* messages are eventually sent and all passive processes change to active mode.

Since at least one correct process has a clock value of $C_{max}(t)$ at the time t the last correct process got up, at least one *(init, $C_{max}(t)$)* message is missed by the initializing process. The first tick, for which it is guaranteed that all correct processes receives at least $f_\ell^{ra} + f_a + f_s + 1$ *(init)* messages, is $C_{max}(t) + 1$. The following Theorem 5.7 gives the latest possible instant when those *(init, $C_{max}(t) + 1$)* messages are received by all correct processes.

Theorem 5.7 (Initialization Time). *Let t be the time when the $n - f_a - f_s - f_o - f_c^{th}$ correct process got up. Progress of the clocks of all correct processes is guaranteed after time $t + \Delta_{init}$, where $\Delta_{init} = 8\tau^+$.*

Proof. Let p be the $n - f_a - f_s - f_o - f_c^{th}$ correct passive process that sends its first *(echo, 0)* message at time t. If there is progress in the system, p will be initialized very quickly because it receives the necessary *(init)* messages τ^+ after they were sent.

If the processes in front are not making progress, the clock value of p at time $t + 2\tau^+$ is $C_p(t + 2\tau^+) \geq C_{max}(t) - 2$. It is reached using the catch-up rule, since by this time the replies to p's *join* message must be received from at least $f_\ell^{ra} + f_a + f_s + 1$ of the most advanced processes (see Lemma 5.6). Process p then sends *(echo, $C_{max}(t) - 2$)*, so that by time $t + 3\tau^+$ every running correct process must have received $n - f_\ell^r - f_a - f_s - f_o - f_c$ *(echo, $C_{max}(t) - 2$)* messages.
Every correct process now sets its clock to $C_{max}(t) - 1$ (if it has not yet done so) and sends an *(init, $C_{max}(t) - 1$)* message. It could be that p does not receive $f_\ell^{ra} + f_a + f_s + 1$ *(init, $C_{max}(t) - 1$)* messages, because there were too many processes that already had the clock value $C_{max}(t) - 1$. So p does not necessarily switch to active mode her.

By time $t + 5\tau^+$, however, all correct processes set their clocks to $C_{max}(t)$ and by time $t + 7\tau^+$ to $C_{max}(t) + 1$. Therefore p and all other correct processes receive at least $f_\ell^{ra} + f_a + f_s + 1$ *(init, $C_{max}(t) + 1$)* messages by time $t + 8\tau^+$ or earlier. □ □

6 From Degraded to Normal Mode - Revisited

The solution discussed in Section 5 has got a major drawback. The maximum overall precision $D_{max} = \lfloor 2\mathcal{P} + \frac{11}{2} \rfloor$ that we must consider is vastly larger than the precision $D_{MCB} = \lfloor \frac{1}{2}\mathcal{P} + \frac{5}{2} \rfloor$ that every correct processes is guaranteed to reach within a very short time after becoming up. The advantage of our solution from Section 5 is the early change into active mode, if there is progress—possibly due to faulty processes—in the system.

This section introduces an alternative solution for the problem of when to switch to active. This switch is only guaranteed to happen later, i.e. when all correct processes are up, even in the case of progress before that.

The algorithm from Figure 2 differs from the previous one from Figure 1 in (1) the fifth **if**, the rule that manages the change to active mode and (2) in the fourth **if**.

Whereas in the previous solution the change happend on the reception of $f_\ell^{ra} + f_a + f_s + 1$ *(init, l)* messages, i.e. the reception of at least one sent by a correct process, it happens in the algorithm of Figure 2 on the reception of $n - f_\ell^r - f_a - f_s - f_o - f_c$ *(echo, l)* messages. Note carefully that it is possible that $l < k$, such that passive processes must keep perception vectors of past rounds. Since this rule requires $n - f_\ell^r - f_a - f_s - f_o - f_c$ messages, it is only guaranteed to be enabled when eventually all correct processes are up. This is done by (1).

Since we do note require that $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ *(init, k)* messages are sent for each round, we can catch-up closer, such that (2) is modified. Using this solution we can reach a smaller D_{max}. We can hence modify our catch-up rule such that the constant factor of D_{MCB} can be decreased by 1.

```
For each correct process

VAR k : integer := 0;
VAR mode : {passive, active} := passive;

if received (init, k) from at least f_ℓ^ra + f_a + f_s + 1 distinct processes
   → send (echo, k) to all;
fi

if received (echo, k) from at least f_ℓ^ra + f_a + f_s + 1 distinct processes
   → send (echo, k) to all;
fi

if received (echo, k) from at least n − f_ℓ^r − f_a − f_s − f_o − f_c distinct processes
   → if mode = active → C := k + 1; /* update clock */
        k := k + 1;
        send (init, k) to all; /* start next round */
fi

/* *** catch-up rule *** */
if received (echo, l) from at least f_ℓ^ra + f_a + f_s + 1 distinct processes
   with l > k
   → if mode = active → C := l; fi /* update clock */
        k := l; /* jump to new round */
        send (echo, k) to all;
fi

/* *** change to active mode *** */
if received (echo, l) from at least n − f_ℓ^r − f_a − f_s − f_o − f_c distinct processes
   → if mode = passive
        → C := max(l + 1, k);
           k := max(l + 1, k);
           mode := active;
           if (l + 1) = k → send (init, k) to all; fi
     fi
fi
```

Figure 2. Clock Synchronization Algorithm for the Hybrid Perception-based Failure Model with Startup Phase and Late Change to Active Mode (Higher Precision)

Since we modified the catch-up rule we can now give better properties for weack synchronization:

Theorem 6.1 (Weak Synchronization Properties). *For $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ and any n_{up}, the algorithm from Figure 2 achieves*

P1WL Weak Correctness. If at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes set their clocks to k by time t, then every correct process sets its clock at least to k by time $t + 2\tau^+$.

P2L Unforgeability. If no correct or benign faulty process sets its clock to k by time t, then no correct process sets its clock to $k + 1$ by $t + 2\tau^-$ or earlier.

P3WL Weak Relay. If a correct process sets its clock to k at time t, then every correct process sets its clock at least to $k - 1$ by time $t + \varepsilon$.

Proof. Similar to proof of Theorem 4.3. Catch-up rule jumps one tick farther. □ □

Precision in degraded mode based on P1WL, P2L and P3WL.

Theorem 6.2 (Precision in Degraded Mode). *In a system with $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ processes, where $0 \leq n_{up} \leq n$ processes, except the faulty ones, are initially synchronized, the algorithm of Figure 1 satisfies the precision requirement (P) with $D_{MCB} = \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor$.*

Proof. Similar to proof of Theorem 4.6. Weak relay just goes back to tick $k-1$ (insted of $k-2$ in Theorem 4.6). □ □

Overall precision:

Theorem 6.3 (Precision). *For $n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$, the algorithm of Figure 2 achieves the precision property (P) during the whole system life-time with $D'_{max} = \lfloor \mathcal{P} + \frac{5}{2} \rfloor$.*

Proof. See proof for Lemma 5.4. k is the smallest clock value—compared to C_{max}—any initializing process is guaranteed to have when changing to active. □ □

Theorem 6.4 (Initialization Time). *Let t be the time when the $n - f_a - f_s - f_o - f_c^{th}$ correct process got up. Progress of the clocks of all correct processes is guaranteed after time $t + \Delta_{init}$, where $\Delta_{init} = 7\tau^+$.*

Proof. See proof for Lemma 5.7. The time all change to active is the time when all correct processes must send *(init, $C_{max}(t) + 1$)* instead of receiving these messages. Hence initialization is done earlier at time $t + 7\tau^+$. □ □

Note that an improved precision also influences the upper envelope bound from Theorem 7.5 in Section 7. Since processes may not jump over D_{max} clock values but only D'_{max}, the upper envelope bound tightens for newly started processes.

7 Clock Synchronization in Normal Mode

We have seen in Theorem 5.7 that progress comes into the system with the last correct process becoming active. In this section we will see the bounds for the accuracy requirement (A). These bounds apply beginning at time t when every correct process p reaches a clock value $C_p(t) > C_{max}(t_{up}) + 1$, t_{up} being the instant the last correct process became up. Such a clock value must be reached using "fresh" messages, i.e. they were not resend during initialization (by the fifth **if**). Hence the time bounds τ^+ and τ^- can be used to calculate the bounds for envelope synchronization.
We now consider a system with $n_{up} = n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ where all correct processes are active. The following two properties from Theorem 7.1 are required in order to give the lower bound from Lemma 7.3. Note that the properties P1W, P2 and P3W from Theorem 4.3 are also guaranteed.

Theorem 7.1 (Clock Synchronization Properties). *For $n_{up} = n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ the algorithm from Figure 1 achieves*

P1F Full Correctness. If all correct processes set their clocks to k by time t, then every correct process sets its clock to $k+1$ by time $t + 2\tau^+$.

P3 Relay. If a correct process sets its clock to k at time t, then every correct process does so by time $t + 2\tau^+ + \varepsilon$.

Proof. Full Correctness. If $k = C_{max}(t)$ all correct processes must have sent *(init, k)* by time t, such that every correct process reaches sufficient evidence by time $t + \tau^+$ to send *(echo, k)* (which they do since all have a clock value k by assumption). By time $t + 2\tau^+$ the perception of every correct process p is $|\mathcal{V}_p^{k'}(t + 2\tau^+)| \geq n - f_\ell^r - f_a - f_s - f_o - f_c$ such that every correct process then sets its clock to $k+1$.
If $k < C_{max}(t)$ at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes must have sent *(echo, k)* by time $t - \tau^-$ (see Lemma 4.2). All correct processes set their clocks to k by time t (by assumption). On the reception of the at least $f_\ell^{ra} + f_a + f_s + 1$ *(echo, k)* messages by time $t + \varepsilon$ all correct processes send *(echo, k)* such that for every correct process p its perception $|\mathcal{V}_p^{k'}(t + \tau^+ + \varepsilon)| \geq n - f_\ell^r - f_a - f_s - f_o - f_c$. Thus all correct processes set their clocks to $k+1$ by time $t + \tau^+ + \varepsilon$.

Relay. Assume $k = C_{max}(t)$. From Weak Relay follows that every correct processes sets its clock to $k-2$ by time $t + \varepsilon$ caused by $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ *(echo, $k-1$)* messages sent by correct processes. These messages force all correct processes to send *(echo, $k-2$)* by time $t + \varepsilon$ which must be received by time $t + \tau^+ + \varepsilon$ by all correct processes which set their clocks to $k-1$. The $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ *(echo, $k-1$)* messages now force all correct processes to send *(echo, $k-1$)* such that by time $t + 2\tau^+ + \varepsilon$ all correct processes set their clocks to k.
If $k < C_{max}(t)$ at least one correct process has already set its clock to $l > k$ at time $t' < t$. In the previous paragraph we have seen that all correct processes must set their clocks to l by time $t' + 2\tau^+ + \varepsilon$ so all correct processes must also set their clocks to k by time $t + 2\tau^+ + \varepsilon$. □ □

Remark Note that our time bound for relay $2\tau^+ + \varepsilon$ is larger than the bound for the original clock synchronization algorithm from [29]. This is due to the fact that our algorithm only sends messages for clock values smaller or equal its local clock. The sending of messages is hence suspended until a process reaches the corresponding clock value.

Lemma 7.2. *Let p be a correct process that sets its clock to k at instant t. p sets its clock to $k+1$ by time $t + 4\tau^+ + \varepsilon$.*

Proof. The proof uses the properties P1F and P3 from Theorem 7.1. P3 (relay) guarantees that all correct processes set their local clocks to k by time $t' = t + 2\tau^+ + \varepsilon$. P1F (full correctness) guarantees that all correct processes, including p, must set their clocks to $k+1$ by time $t' + 2\tau^+$. Hence p sets its clock by time $t + 4\tau^+ + \varepsilon$. □ □

Lemma 7.3 (Lower Envelope Bound). *For the described system with $n_{up} = n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ and all correct processes are active, $\frac{t_2 - t_1}{2\tau^+} - 4 + \frac{1}{\mathcal{P}} < C_p(t_2) - C_p(t_1)$ for all correct active processes p at all times $t_2 \geq t_1$.*

Proof. Let $C_p(t_1) = k$ and $C_p(t_2) = k + l$. p has set its clock to k at instant $\sigma^k \leq t_1$ and to $k+l$ at instant $\sigma^{k+l} \leq t_2$. Hence $t_2 - t_1 < \sigma^{k+l+1} - \sigma^k \leq \sigma^{k+l} - \sigma^k + 4\tau^+ + \varepsilon$ (following Lemma 7.2).
From Relay follows that process p may wait as long as $2\tau^+ + \varepsilon$ for the other correct processes to catch-up to its round and force progress in p's clock value. Hence the time there is progress without waiting is $t_2 - t_1 - 2\tau^+ - \varepsilon < t_2 - t_1$. Such that $t_2 - t_1 - 6\tau^+ - 2\varepsilon \leq \sigma^{k+l} - \sigma^k$. ($t_2 - t_1 - 6\tau^+ - 2\varepsilon = t_2 - t_1 - 8\tau^+ + 2\tau^-$).
$C_p(t_1) = C_p(\sigma^k)$ and $C_p(t_2) = C_p(\sigma^{k+l})$ such that the number of steps $l = C_p(\sigma^{k+l}) - C_p(\sigma^k)$. l is always larger than the smallest possible number of steps within any time interval of the size $t_2 - t_1 - 8\tau^+ + 2\tau^-$. Hence $\frac{t_2 - t_1}{2\tau^+} - 4 + \frac{1}{\mathcal{P}} < C_p(t_2) - C_p(t_1)$ (following full correctness). □ □

A straight forward approach for giving the upper bound for the envelope condition would be to give the shortest possible interval between clock updates like Lemma 4.4. Lemma 4.4, however, only refers to the most advanced clock values. Due to the catch-up rule no shortest possible interval between consecutive clock updates can be given for clocks that are behind. In Lemma 7.4 we use the precision of the clocks in conjunction with Lemma 4.4 to get an upper envelope bound.

Lemma 7.4 (Upper Envelope Bound). *For the described system with $n_{up} = n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ and all correct processes are active, $C_p(t_2) - C_p(t_1) < \frac{t_2 - t_1}{2\tau^-} + D_{max} + 1$ for all correct active processes p at all times $t_2 \geq t_1$.*

Proof. From Theorem 5.5 (precision) follows that $C_{max}(t) - D_{max} \leq C_p(t) \leq C_{max}(t)$ at all times t for all correct processes p. Specifically at instant t_1 the clock value of any correct process p is therefore bounded by $C_{max}(t_1) - D_{max} \leq C_p(t_1)$, and at instant t_2 there is an upper bound of $C_p(t_2) \leq C_{max}(t_2)$. Thus $C_p(t_2) - C_p(t_1) \leq C_{max}(t_2) - C_{max}(t_1) + D_{max}$.
$C_{max}(t_2) - C_{max}(t_1)$ needs to be bounded: Let $C_{max}(t_1) = k$ and $C_{max}(t_2) = k + l$. C_{max} is set by any correct process to k at instant $\sigma^k \leq t_1$ and to $k+l$ at instant $\sigma^{k+l} \leq t_2$. Hence $t_2 - t_1 > \sigma^{k+l} - \sigma^{k+1} \geq \sigma^{k+l} - \sigma^k - 2\tau^-$ (following unforgeability) and directly follows $t_2 - t_1 + 2\tau^- > \sigma^{k+l} - \sigma^k$.
$C_{max}(t_1) = C_{max}(\sigma^k)$ and $C_{max}(t_2) = C_{max}(\sigma^{k+l})$ such that the number of steps $l = C_{max}(\sigma^{k+l}) - C_{max}(\sigma^k)$. l is always smaller than the largest possible number of steps within any time interval of the size $t_2 - t_1 + 2\tau^-$. Using Lemma 4.4 we get $C_{max}(t_2) - C_{max}(t_1) < \frac{t_2 - t_1}{2\tau^-} + 1$. Thus $C_p(t_2) - C_p(t_1) < \frac{t_2 - t_1}{2\tau^-} + D_{max} + 1$. □ □

Theorem 7.5 (Envelope Condition). *The described system with $n_{up} = n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ and all correct processes being active, satisfies the following envelope condition*

$$\frac{t_2 - t_1}{2\tau^+} - 4 + \frac{1}{\mathcal{P}} < C_p(t_2) - C_p(t_1) < \frac{t_2 - t_1}{2\tau^-} + D_{max} + 1$$

for all correct active processes p for all times $t_2 \geq t_1$.

Proof. See Lemma 7.3 and Lemma 7.4. □ □

Remark As we have seen in Lemma 5.2, there is a time for each correct processes from when it is guaranteed to remain within precision D_{MCB} to all other correct processes and hence stays in precision with C_{max}. Note that our upper envelope bound from Theorem 7.5 uses D_{max}. That is because every active correct processes could jump over D_{max} clock values once, shortly after it became active. During normal operation, however, when all correct processes are within precision D_{MCB}, the upper envelope bound tightens from $\frac{t_2 - t_1}{2\tau^-} + D_{max} + 1$ to $\frac{t_2 - t_1}{2\tau^-} + D_{MCB} + 1$. Unfortunately a correct processes cannot be guaranteed to determine the time when this happens and for safety reasons the bound containing D_{max} must be considered.

We have seen that progress is guaranteed if all correct processes are eventually active. This is enforced by the third **if**. This led us to the relay property from Theorem 7.1. Compared to weak relay from Theorem 4.3 relay is on the one hand better regarding the clock values processes are guaranteed to reach. On the other hand it is worse regarding the time bound when this happens. Our precision bound D_{MCB} from Theorem 4.6 is built upon weak relay. In the following Theorem 7.6—which we build upon relay—we will examine an additional bound for precision D'_{MCB} that is only guaranteed to hold if all correct processes are active.

Theorem 7.6 (Precision of MCB during normal operation). *For the algorithm from Figure 1 there exists a constant D'_{MCB} such that $\mid C_p(t) - C_q(t) \mid \leq D'_{MCB}$ for all processes p and q at every time t for $n_{up} = n \geq 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ where all correct processes are active. $D'_{MCB} = \lfloor \frac{3}{2}\mathcal{P} + \frac{1}{2} \rfloor$.*

Proof. Assume a correct process p has a local clock value $k \geq 0$ within an unknown precision D'_{MCB} to all other correct processes—and therefore also to $C_{max}(t')$—at real-time t'. We now use relay (P3), Definition 7 and Lemma 4.5 to reason about D'_{MCB} by calculating $C_{max}(t)$ for a time $t > t'$.

Let p advance its clock to $k+1$ such that $\sigma_p^k = t > t'$. Process p has not done so before t because no other correct process has done so before $t - 2\tau^+ - \varepsilon$ following directly from relay (P3), thus $\sigma_{first}^k \geq t - 2\tau^+ - \varepsilon$.

From Lemma 4.5 follows that $C_{max}(t) \leq \lfloor \frac{t-(t-2\tau^+-\varepsilon)}{2\tau^-} \rfloor + I_\sigma(t-\varepsilon) + C_{max}(t-\varepsilon)$. Let us now take a closer look at the term $I_\sigma(t-2\tau^+-\varepsilon) + C_{max}(t-2\tau^+-\varepsilon)$: If $\sigma_{first}^k = t - 2\tau^+ - \varepsilon$ and therefore $t - 2\tau^+ - \varepsilon$ is synchronized with C_{max}, $C_{max}(t-2\tau^+-\varepsilon) = k+1$ and $I_\sigma(t-2\tau^+-\varepsilon) = 0$ (following Definition 7). If on the other hand $\sigma_{first}^k > t - 2\tau^+ - \varepsilon$, $C_{max}(t-2\tau^+-\varepsilon) = k$ and $I_\sigma(t-2\tau^+-\varepsilon) = 1$. In both cases $I_\sigma(t-2\tau^+-\varepsilon) + C_{max}(t-2\tau^+-\varepsilon) = k+1$ such that $C_{max}(t) \leq \lfloor \frac{2\tau^+ + \varepsilon}{2\tau^-} \rfloor + k + 1$ thus $C_{max}(t) \leq \lfloor \frac{3}{2}\mathcal{P} + \frac{1}{2} \rfloor + k$.

Process p has a clock value $C_p(t') = k$ at a time $t' < t$ which is by assumption within precision. Since $C_p(t') < C_p(t)$ and $C_{max}(t') \leq C_{max}(t)$, we get a bound for our precision D'_{MCB} from the difference $C_{max}(t) - C_p(t') = C_{max}(t) - k \leq \lfloor \frac{3}{2}\mathcal{P} + \frac{1}{2} \rfloor$. □ □

When comparing our two bounds on precision—which are both valid during normal operation—we discover a "break even" at $\mathcal{P} = 2$. Only for very small uncertainties ($\tau^+ < 2\tau^-$) D'_{MCB} and hence the third **if** guarantees a better precision. In systems with larger uncertainties our first bound D_{MCB}—enforced by the fourth **if**—is better. But note that due to initialization requirements our algorithm has a larger time bound on relay than the classic clock synchronization algorithm from [29] which would guarantee a precision of $D_{CB} = \lfloor \mathcal{P} + \frac{1}{2} \rfloor$[10].

8 Conclusions

We described and analyzed a clock synchronization algorithm for partially synchonous systems with unknown bounds $[\tau^-, \tau^+]$ upon delays, which works during system startup and tolerates a large number of hybrid process and link failures. Whereas accuracy (and hence progress of the clocks) can only be guaranteed when sufficiently many correct processes are eventually up and running, it guarantees bounded precision D_{max} during both startup and normal operation. Our clock algorithm is hence a promising basis for studying other partially synchronous algorithms during system startup.

Our algorithm is actually a purely time- and timer-free and hence "asynchronous" algorithm. Its timing properties hence solely "immerse" [8] from the underlying system, which means that e.g. its precision D_{max} adapts to the actual τ^+ and τ^-. Even more, its precision actually depends only upon the ratio $\mathcal{P} = \tau^+/\tau^-$. In terms of assumption coverage, this is an advantageous property: At heavy network load, when messages have to be queued at the network interfaces, both τ^+ and τ^- increase. Therefore it is possible that an assumption on $\mathcal{P}$ holds despite of the fact that an assumption on τ^+ does not. In this case our algorithm would hence still provide the expected performance, whereas a synchronous would fail.

References

[1] M.H. Azadmanesh and Roger M. Kieckhafer. Exploiting omissive faults in synchronous approximate agreement. *IEEE Transactions on Computers*, 49(10):1031–1042, October 2000.

[10] The proof for D_{CB} is identical to the proof of Theorem 7.6. For the time bound on relay a smaller value ($\tau^+ + \varepsilon$ instead of $2\tau^+ + \varepsilon$) must be considered.

[2] Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, 1999.

[3] Danny Dolev, Joseph Y. Halpern, and H. Raymond Strong. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Sciences*, 32:230–250, 1986.

[4] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[5] Christof Fetzer and Flaviu Cristian. An optimal internal clock synchronization algorithm. In *Proceedings 10th Annual IEEE Conference on Computer Assurance*, Gaithersburg, MD, June 1995.

[6] Eli Gafni. Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 143–152. ACM Press, 1998.

[7] Jim N. Gray. Notes on data base operating systems. In G. Seegmüller R. Bayer, R.M. Graham, editor, *Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, chapter 3.F, page 465. Springer, New York, 1978.

[8] J.-F. Hermant and Gerard Le Lann. Fast asynchronous uniform consensus in real-time distributed systems. *IEEE Transactions on Computers*, 51(8):931–944, August 2002.

[9] Roger M. Kieckhafer, Chris J. Walter, Alan M. Finn, and Philip M. Thambidurai. The MAFT architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37:398–405, April 1988.

[10] Gérard Le Lann and Ulrich Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003. (submitted).

[11] Barbara Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, 6:211–219, 1993.

[12] Jennifer Lundelius-Welch and Nancy A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.

[13] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[14] Paul S. Miner. Verification of fault-tolerant clock synchronization systems. *NASA Technical Paper 3349*, November 1993.

[15] Stephen Ponzio and Ray Strong. Semisynchrony and real time. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG'92)*, pages 120–135, November 1992.

[16] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, October 1990.

[17] John Rushby. A formally verified algorithm for clock sychronization under a hybrid fault model. In *Proceedings ACM Principles of Distributed Computing (PODC'94)*, pages 304–313, Los Angeles, CA, August 1994.

[18] Ulrich Schmid. Interval-based clock synchronization with optimal precision. Technical Report 183/1-78, Technische Universität Wien, Department of Automation, July 1997. (to appear in Information and Computation).

[19] Ulrich Schmid, editor. *Special Issue on The Challenge of Global Time in Large-Scale Distributed Real-Time Systems*, Real-Time Systems 12(1–3), 1997.

[20] Ulrich Schmid. Orthogonal accuracy clock synchronization. *Chicago Journal of Theoretical Computer Science*, 2000(3):3–77, 2000.

[21] Ulrich Schmid. How to model link failures: A perception-based fault model. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, pages 57–66, Göteborg, Sweden, July 1–4, 2001.

[22] Ulrich Schmid and Christof Fetzer. Randomized asynchronous consensus with imperfect communications. Technical Report 183/1-120, Department of Automation, Technische Universität Wien, January 2002. (submitted).

[23] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, March 1997.

[24] Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing*, 14(2):101 – 111, May 2001.

[25] Ulrich Schmid and Bettina Weiss. Synchronous Byzantine agreement under hybrid process and link failures. Technical Report 183/1-124, Department of Automation, Technische Universität Wien, November 2002. (submitted; replaces TR 183/1-110).

[26] Ulrich Schmid, Bettina Weiss, and John Rushby. Formally verified byzantine agreement in presence of link faults. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 608–616, July 2-5, 2002.

[27] Fred B. Schneider. Understanding protocols for byzantine clock synchronization. Technical Report 87-859, Cornell University, Department of Computer Science, August 1987.

[28] Barbara Simons, Jennifer Lundelius-Welch, and Nancy Lynch. An overview of clock synchronization. In Barbara Simons and A. Spector, editors, *Fault-Tolerant Distributed Computing*, pages 84–96. Springer Verlag, 1990. (Lecture Notes on Computer Science 448).

[29] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[30] T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.

[31] Wilfried Steiner and Michael Paulitsch. The transition from asynchronous to synchronous system operation: An approach for distributed fault-tolerant systems. *Proceedings of the The 22nd International Conference on Distributed Computing Systems*, July 2002.

[32] Paulo Veríssimo, Antonio Casimiro, and Christof Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Proceedings IEEE International Conference on Dependable Systems and Networks (DSN'01 / FTCS'30)*, pages 533–542, 2000.

[33] Chris J. Walter and Neeraj Suri. The customizable fault/error model for dependable distributed systems. *Theoretical Computer Science*, 290:1223–1251, October 2002.

[34] Josef Widder. Switching On: How to boot clock synchronization in partially synchronous systems. Technical Report 183/1-125, Department of Automation, Technische Universität Wien, December 2002. (submitted).