Distributed Computing manuscript No.
(will be inserted by the editor)

Technical Report 183/1-126, Dept. of Automation, TU Vienna, January 2003

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

Josef Widder, Ulrich Schmid

Technische Universität Wien,
Embedded Computing Systems Group (E182/2),
Treitlstrasse 3, A-1040 Vienna,
e-mail: {widder,s}@ecs.tuwien.ac.at

The date of receipt and acceptance will be inserted by the editor

Summary. This paper provides description and analysis of 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. Detailed formulas for resilience, precision and envelope bounds of our algorithm are also determined.

Key words: Fault-Tolerant Distributed Algorithms – System Startup – Initial Clock Synchronization – Hybrid Failure Models – Link Failures

1 Introduction

Clock synchronization is an important service in distributed systems, see [17] 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. We will concentrate on deterministic clock synchronization, which must guarantee the following properties:

(P) *Precision*: Any two correct clocks may deviate at most by some D_{max}.
(A) *Accuracy*: Any correct clock remains within a linear envelope of real-time.

Many different clock synchronization algorithms have been proposed in literature; see [22,24,34] for an overview. Most of these algorithms assume systems with known bounds on computation step times and communication delays, 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 processes may be lost, and failure assumptions like the one that at most one third of the processes are Byzantine faulty—which is necessary [4] for achieving (P) and (A) in presence of Byzantine failures—do not hold. Moreover, many real networks (like the Internet) cannot be modeled properly as synchronous system [2, 11, 38]. To implement clock synchronization[1] in such systems, a time and timer free startup mechanism is required.

In [40], we provided a solution to this problem that—unlike naive startup algorithms—avoids increasing the required number of processes and/or adding a priori assumptions to the system model: By modifying the clock synchronization algorithm by Srikanth and Toueg [35], which is based upon the well-known *consistent broadcasting* primitive, we provided an algorithm that is completely time- and timer-free and needs just $n \geq 3f + 1$ 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 constant precision D_{max} — which depends upon these bounds — during the whole system operation. Progress of the clocks, i.e., accuracy, is only guaranteed when sufficiently many correct processes are eventually up and running.

* This research is supported by the Austrian START program Y41-MAT, in the context of our W2F-project *http://www.auto.tuwien.ac.at/Projects/W2F/*
Correspondence to: Josef Widder

[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 4 for details.

In this paper, we will present and completely[2] analyze a hybrid version of the algorithm of [40] under an extension of the *perception based failure model* introduced in [27]. Our new algorithm not only provides considerably improved fault-tolerance with respect to process failures, but shows that a large number of link failures can be tolerated easily as well. It can therefore be applied even in typical wireless settings, where link failure rates up to 10^{-2} are common [31]. Moreover, detailed formulas on how many processes are required in order to tolerate given failure types make it easy to dimension a system with given failure probabilities for various failure types. Note also that decreasing system size increases the overall dependability since less components can fail. Clock synchronization and, in turn, all algorithms that depend upon synchronized clocks can hence safely be implemented in systems with hybrid node and link failures, despite of unpredictable startup times.

The remainder of this paper is organized as follows: After a survey of existing work in Section 2, we extend the perception based failure model of [27] to the system startup phase in Section 3. In Section 4, we present our hybrid algorithm and its operation principles. Section 5 and Section 6 contains the analysis of our algorithm in early and degraded mode, respectively, when not sufficiently many processes have completed booting. In Section 7, we investigate our algorithm's performance in normal mode, when sufficiently many processes are eventually up and running.

2 Related Work

Although clock synchronization in synchronous systems [4, 20, 22, 24, 33, 34], as well as in partially synchronous systems [6, 21], is a very well-researched field, there are only a few papers [12, 18, 20, 35, 37] 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 [37], which deals with a custom solution to the startup problem in the very specific TTP system architecture, and [12], which considers an approach for initialization in the MAFT architecture under stronger system assumptions. None of these solutions is time(r)-free and works in partially synchronous systems. This is also true for self-stabilizing byzantine clock synchronization algorithms [3, 5]. They are an overkill for solving the booting problem as well. We believe that therefore none of the existing work guarantees constant stabilization times like we do for booting.

Clock synchronization in presence of link failures has been studied in [23] and in part of our previous work [15, 25, 26, 29, 30]. None of these papers considered initial startup, however. Moreover, with the exception of [15, 26], all those papers consider synchronous systems only. In [15], we introduced a time(r)-free implementation of a perfect failure detector in partially synchronous systems based upon consistent broadcasting. In [16], we provided a hybrid variant of this failure detector that works under the perception based failure model of [27]. In both [15, 16], however, we assumed that all processes are up right from the start.

[2] Several proofs had to be omitted in [40] due to lacking space.

3 Perception Based Failure Model

This section contains an overview of our failure model, which extends the perception based model introduced in [27] by adding messages with history and proper handling of the startup phase. 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, 39], 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 lack of space, we will omit a detailed 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 [14] in real systems, see [31] for an example.

The physical failure model can be mapped to a more abstract (and vastly simpler) *perception failure model*, which is similar in spirit to the round-by-round fault detector approach of [8]. It is a generalization of the synchronous model of [31, 32], 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.

3.1 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. Links need not necessarily provide FIFO transmission. The system will execute a distributed round based algorithm made up of one or more concurrent *processes* at every processor. Any two processes at different processors can communicate bidirectionally with each other via the interconnecting links. Every processor is identified by a unique *processor id* $p \in \Pi = \{1, \ldots, n\}$; every process is uniquely identified system-wide by the tuple (processor id, process name), where the *process name* N is chosen from a suitable name space. Since a process will usually 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. Our model, however, allows sender and receiver process to have different names as well.

Since we restrict our attention to round based algorithms, 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* $\mathcal{R}_p^k \subseteq \{1, \ldots, n\}$.[3] We assume that every (non-faulty) receiver q knows its current *sender set* $\mathcal{S}_q^k = \{p : q \in \mathcal{R}_p^k\}$ 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. Note that this convention does not prohibit an efficient direct implementation of self-reception, provided that the resulting end-to-end transmission delay is taken into account properly.

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[4] 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. In case of multiple round l messages from the same sender q, which must be faulty here, the receiver could also drop all messages and set $V_p^{q,l}$ to some obviously faulty value, instead of retaining the value from the first message.

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 $k+1$. Note that round switching is event based—and part of the particular algorithm—in case of asynchronous systems but enforced externally in case of synchronous systems. At the round switching time, the value $V_p^{k+1} = F_p\big(\mathcal{V}_p^k(\sigma_p^k), \Sigma_p\big)$ 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 $\mathcal{V}_p^k = \mathcal{V}_p^k(\sigma_p^k)$ and p's local state Σ_p at time σ_p^k.

The above execution model is generalized by introducing messages with arbitrary history size $0 \le h \le \infty$. 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. Note that such "late" perceptions are retained even when a proper round l message drops in later (which could occur since we do not assume FIFO channels); unlike the arrival of two proper round k messages, this does not constitute an error.

Although round l perceptions filled in upon reception of a round $k' > l$ message are usually time faulty, in the sense that they should have arrived earlier (namely, in the process's proper 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 `line 18` resp. `line 25` the algorithm of Figure 4, for example, we employ $h = 1$ for *echo*-messages to substitute round l messages that were not received due to late booting and catch-up. In case of this particular algorithm, "late" perceptions need not be considered time faulty. Typically, however, a history size $h > 0$ must be accounted for in the failure model, see Section 3.3.

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 failure) 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 the local round k computation $V_p^{k+1} = F_p\big(\mathcal{V}_p^k(\sigma_p^k), \Sigma_p\big)$ is also included in $\delta_q^{p,k}$ (or in $\delta_q^{p,k+1}$), which creates the abstraction of algorithms that consist of statements executed in zero time.

Figure 1 shows this relation. Note carefully that $\delta_q^{p,k}$ does not only depend upon the local computation and network load related to the particular round k message sent from p and q. As argued in [15,16], scheduling issues make $\delta_q^{p,k}$ dependent upon any round k message that is sent and received by p and q.

Our model stipulates minimum and maximum values $\tau^- > 0$ and $\tau^+ < \infty$[5], not necessarily known to the algorithm, such that

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

[3] 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.

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

[5] Of course, our model would also allow the setting $\tau^- = 0$ and/or $\tau^+ = \infty$, but this would result in $\Theta = \tau^+/\tau = \infty$ and is hence ruled out in this paper.

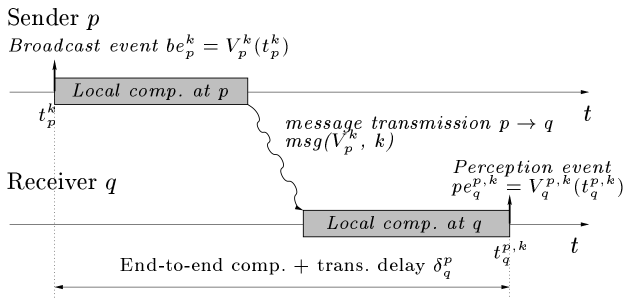

Fig. 1. *Relation between process p's round k broadcast event* $be_p^k = V_p^k(t_p^k)$ *and its corresponding perception event* $pe_q^{p,k} = V_q^{p,k}(t_q^{p,k})$ *at process q.*

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.[6] 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. $\Theta = \tau^+/\tau^-$.

The above description of the execution model of a single process must still be extended to multiple processes. Three functions and one additional event are sufficient for this purpose:

- *create*(P) creates and initializes a process with name P on the caller's processor, including its perception vector and reception processing setup. It is usually called at boot time, for any process that will eventually run on a certain processor. If called at time t_q, it raises a parameterless init event ie_q^0 occurring at time $\iota_q^0 \geq t_q$ that signals completion of booting. It ensures that every correct process p's broadcast event that occurs at time $t_p^k \geq \iota_q^0 - \tau^-$ generates a perception event $pe_q^{p,k}$ at q at some time $t_q^{p,k} \geq \iota_q^0$, provided that the message is not hit by a link failure. Earlier broadcast event's are usually completely lost, however, in the sense that there is no corresponding perception event at q.
- *init*$(P, k, V_p^k, \mathcal{R}_p^k)$ raises the broadcast event be_p^k with receiver set $\mathcal{R}_p^k$ in process P at the caller's processor p. It can be called by any process running on the same processor as P and is also used internally by P to initiate be_p^k at the round switching time σ_p^{k-1} in round $k > 0$. Obviously, *init* must not be called more than once and only after ι_p^0.
- *wait*(P, k, X) blocks the caller process Q until process $P \neq Q$ on Q's processor has completed round k; a value X from P is optionally returned to the caller upon unblocking. Note that this operation does not affect process P's execution and may require some memory if P terminates round k before *wait* is called.

In the pseudo code employed for the description of our algorithms, a more high-level way of expressing concurrency will usually be used:

cobegin
  { block_1 }
  ⋮
  { block_k }
coend

is a concise description of k concurrent processes, each executing one block_i.

3.2 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. Note carefully that we do not even allow spurious messages from a down process or any of its links here. Correct processes boot at unknown times that cannot be bounded a priori.

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

1. *down*: A process remains down when it has not been created yet or has not completed booting. Messages that drop in while a process is down are usually completely lost, i.e., do not generate a corresponding perception event.
2. *up*: A process q gets up when it has completed booting, at time ι_q^0. Recall that every correct process p's broadcast event that occurs at time $t_p^k \geq \iota_q^0 - \tau^-$ generates a perception event $pe_q^{p,k}$ at q at some time $t_q^{p,k} \geq \iota_q^0$, provided that the message is not hit by a link failure.
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 typically sends a special *join* message to all its peers. The first reception of a *join* message from some process p causes the receiver to send an algorithm-dependent reply message to p (point-to-point); subsequent *join* messages from the same sender are usually ignored.
4. *active*: A process that has completed its initialization phase is called active.

In case of the algorithm of Figure 4, for example, a passive process broadcasts *join*—namely, *(init,* 0*)*, in order to get the last *(init, k)* and *(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}.

The communication pattern for joining is obviously different from ordinary rounds. Whereas *join* is broadcast by a single joining process p, the replies are typically sent back by all peers via point-to-point messages.

[6] Those assumptions can be relaxed: The Θ-model of [15] allows τ^- and τ^+ to vary with time, and the typically very small self-reception delay could be considered as an (early) link timing failure and hence be masked by increasing f_ℓ^{ra} and f_ℓ^{sa} by 1.

All the processes' *join* can be considered as a special booting round's messages, however, so both execution model and failure model apply to *join* as for any round k message. Response messages are different, however. Although compatibility with our execution model is maintained by assuming $\mathcal{R}_q^{resp} = \{p\}$ and $\mathcal{S}_p^{resp} = \Pi$, we need some additional assumptions with respect to our failure model.

Since the semantics of response messages are algorithm dependent, it does not make sense to define a generic model here. Basically, the joiner p must be provided with sufficient information to construct an internal state that is equivalent—in some algorithm dependent sense—to the state it would have had if it had not missed earlier messages due to late booting. This is a demanding requirement, however, since e.g. failures that occurred before p got up must be taken into account.

In case of the clock synchronization algorithm of this paper, for instance, any process s that successfully received p's *join* responds by retransmitting[7] the last sent *(init, k_s)* and *(echo, k'_s)*. Retransmissions must happen in a way, however, which lets the resulting perception matrices maintain compliance with our failure model. For example, if arbitrary link failures in process s's original round k_s broadcast already caused f_ℓ^{ra} receivers $q \neq p$ to deliver faulty round k_s perceptions V_q^{s,k_s}, it must not happen that s's response message to p's *join* suffers from an arbitrary link failure as well. Response messages may hence be viewed as messages belonging to the original round k broadcast here, which are deferred until the appropriate receiver is up. As in case of messages with history $h > 0$, perceptions recovered via response messages are usually time faulty, however (see Remark 4 on Definition 4).

3.3 Physical Failure Model

Our physical failure model basically[8] assumes that at most f_a, f_s, f_o and f_c processors may be arbitrary, symmetric, omission, and clean crash faulty during any single round. Clean crash faulty processes may be mute to all receiving processes (consistently) in some round, omission faulty processes may fail to send the correct message to an arbitrary subset of the receivers (inconsistently). Symmetric failures allow a process to disseminate an erroneous value consistently, whereas arbitrary faulty processes may even exhibit Byzantine behavior.

Since we will often establish *uniform* results (which also hold for processors with benign failures, namely, omission and clean crash faulty processes unless they have crashed), we will use the term *obedient* process to denote non-faulty, omission, and clean crash faulty processes that are still alive at the point of observation.

As far as link failures are concerned, *every* receiving process may experience at most f_ℓ^r link failures affecting different peers in the reception of *every* round, with at most $f_\ell^{ra} \leq f_\ell^r$ arbitrary time/value faulty ones (producing early or late message receptions and/or faulty values). Similarly, *every* sending process may generate at most f_ℓ^s link failures when sending a message to all its peers per round, with at most $f_\ell^{sa} \leq f_\ell^s$ arbitrary time/value faulty ones among those. Assigning different numbers of send and receive link failures is useful for modeling certain restricted process failures [31]; usually, however, $f_\ell^r = f_\ell^s = f_\ell$ and $f_\ell^{ra} = f_\ell^{sa} = f_\ell^a$. As long as processes do not exceed those limits, they need not be considered (process) faulty. Further details will be provided in the next section.

3.4 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 $|\mathcal{V}_q^k(t + \Delta t)| \geq |\mathcal{V}_q^k(t)|$ for any $\Delta t \geq 0$, 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, ultimately, 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 3.1), 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 *perception failure model* is much simpler than the physical one and therefore more suitable for analyzing an algorithm's fault-tolerance properties.

Our formalization hence rests upon the $n \times n$ *perception matrix* $\mathcal{V}^k(t)$[9] of round k perceptions observed at the same arbitrary real-time t—typically, some process's round switching time—at all processes:

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

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:

[7] Note that those retransmissions may cause the arrival of two round k messages from the same sender, which is usually considered an error in our execution model, recall Section 3.1. This is of course not true here, so we just assume that the first perception entered into some round k perception vector is retained—and not overwritten by some error value—when a second round k message drops in.

[8] In our fully-fledged physical failure model, we actually differentiate between the number of physical failures $\phi_{av}, \phi_{sv}, \ldots$ and the number of perceptions failures $f_a, f_s, \ldots$ for generality. More specifically, formulas like $f_a = \phi_{av}$ for the perception failure numbers, given the physical failure numbers, are provided when mapping the the physical failure model to the perception failure model.

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

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:*

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

Next, we have to classify sender[10] process failures. This requires the important notion of *obedient* processes: An obedient process is an alive process that faithfully executes the particular algorithm. It gets its inputs and perform its computations exactly as a non-faulty process, but it might fail in specific ways to communicate its value to the outside world. We will subsequently use this term instead of non-faulty whenever a process acts as a receiver ("obedient receiver"), since this will allow us to reason about the behavior of (benign) faulty processes in case of uniform properties [10] as well. If $\mathcal{R}_p$ denotes some sender p's receiver set, let $\overline{\mathcal{R}}_p \subseteq \mathcal{R}_p$ denote the set of obedient processes among those.

Whereas the physical failure model differentiates timing failures according to $\delta_q^{p,k} \in \boldsymbol{\tau}$ vs. $\delta_q^{p,k} \notin \boldsymbol{\tau}$ and hence incorporates those quantities explicitly, it is solely the choice of t that is used in Definition 2 for this purpose: It only depends upon t whether a non-faulty perception V_q^p represents a perception event pe_q^p from a non-faulty or rather a timing faulty process p. Hence, neither $\delta_q^{p,k}$ nor $\boldsymbol{\tau}$ will show up in the definitions of the perception failure model below.

Definition 2 (Perception Process Failures). *Let p be a (faulty) sender process and g be some obedient receiver with $\emptyset \neq V_g^p \in \mathcal{V}(t)$, if there is any such g. 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 obedient receivers $q \in \overline{\mathcal{R}}_p \subseteq \mathcal{R}_p$ as follows:*

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

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

The following Definition 3 specifies the possible failures in perceptions caused by link failures. Note that it is formalized for non-faulty sender processes only, although link failures may of course also hit faulty senders in our final failure model.

Definition 3 (Perception Link Failures). *In the absence of sender process failures, a failure of the link from sender p to an obedient receiver q is 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.

To overcome the impossibility of consensus in presence of unrestricted link failures [9, 19], it turned out that send and receive link failures should be considered independently [31, 32]. The following link-failure-related parameters are hence incorporated in the final perception failure model of Definition 4 below:

($\mathrm{A1}^s$) *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 caused by link failures, see Figure 2.

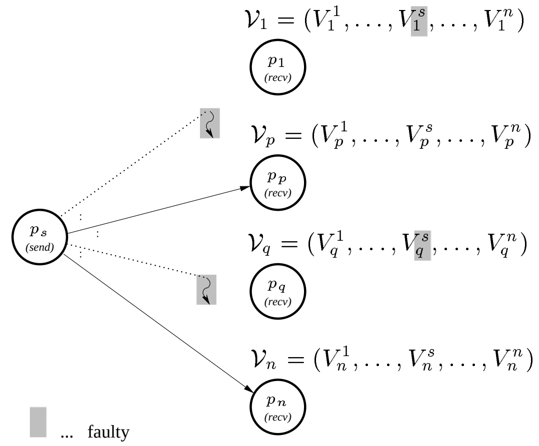

Fig. 2. *Example of a broadcast failure that affects the messages of two recipients.*

($\mathrm{A1}^r$) *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 caused by link failures, see Figure 3.

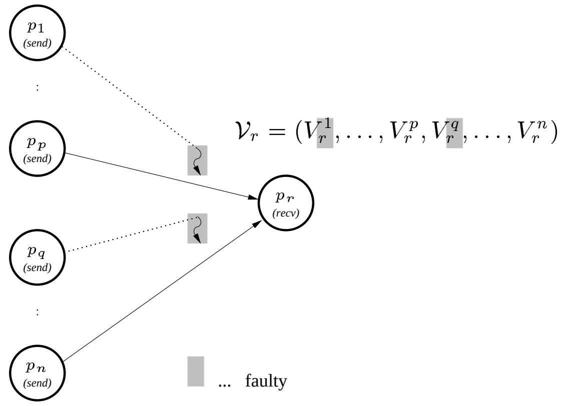

Fig. 3. *Example of a receive failure that involves the messages from two senders.*

[10] Receiver process failures will be considered below when introducing link failures.

Separating broadcast and receive link failures makes sense due to the fact that we consider the unidirectional channels, rather than the bidirectional links, as single fault containment regions: Broadcast link failures affect outbound channels, whereas receive link failures affect inbound channels. Still, broadcast and receive link failures are of course not independent of each other: If a message from process p to q is hit by a failure in p's message broadcast, it obviously contributes a failure in process q's message reception as well. Nevertheless, our failure model considers (A1^s) and (A1^r) as independent of each other and of process failures, for any process in the system and any broadcast/reception. Only the model parameters f_ℓ^s and f_ℓ^r cannot be independently chosen (without restricting the link failure patterns), since the system-wide number of broadcast and receive link failures must of course match. Hence, $f_\ell^s = f_\ell^r = f_\ell$ is the most natural choice, although other settings can also be considered [31].

Note carefully that we allow *every* process in the system to commit up to f_ℓ^s broadcast and up to f_ℓ^r receive link failures in every round, without considering the process as faulty in the usual sense. In addition, the particular links actually hit by a link failure may be different in different rounds. A process must be considered (omission) faulty, however, if it exceeds its budget f_ℓ^s of broadcast link failures in some round. Note that a process that experiences more than f_ℓ^r receive link failures in some round must usually be considered (arbitrary) faulty, since it might be unable to correctly follow the algorithm after such an event.

The following Definition 4 contains our complete perception based failure model, which just specifies the properties of any round's perception matrix $\mathcal{V}^R(t)$. Note carefully that this definition is valid for arbitrary times t, including those where the perceptions from some senders did not yet arrive (and are hence $\emptyset$).

Definition 4 (Asynchronous Perception Failure Model). *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 obedient 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 correspond to arbitrary, symmetric, omission, crash, and manifest faulty processes and may hence contain perceptions $V_q^{p,k}$ according to Definition 2.

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

(A1^r) In every single row q corresponding to an obedient 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 ones obtained in the absence of receive link failures. At most $f_\ell^{ra} \le f_\ell^r$ of those may be link arbitrary faulty.

(A2) Process q can be sure about 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 obedient receivers q and r via non-faulty links.

Remarks:

1. The effects of process failures (P1) and link failures (A1^s), (A1^r) are considered orthogonal; it can hence happen that a link failure hits a perception originating from a faulty sender process. This is also true for manifest and clean crash failures, where link arbitrary failures could create non-empty perceptions at some receivers.
2. (P1) implies that our perception failure model is compatible with traditional process failure models. For example, f_a Byzantine faulty processes can generate inconsistently faulty perceptions only in the f_a columns they correspond to. Hence, all existing lower bounds and impossibility results, like $n \ge 3f_a + 1$ for consensus in presence of f_a arbitrary process failures [13], remain valid.
3. Analysis results under our perception failure model are automatically valid for partially connected networks as well, since the consequences of an incomplete communication graph can be viewed as (persistent) link omission failures. However, sparsely connected graphs and, in particular, partitioned ones cannot be modeled, cp. [31, Thm.3.2].
4. Our failure model also allows arbitrary resp. symmetric faulty processes to commit an early/late timing failure inconsistently resp. consistently (within ε). Neither omission faulty nor clean crash faulty processes may generate time-faulty perceptions in case of no history ($h = 0$). In case of messages with history or responses to *join*-messages, however, the latter cannot usually be guaranteed: Just take an omission faulty process s, for example, which omits sending a round k message to p but not to q. If this process sends a round $k+1$ message to p in the next round, it also generates a round k perception $V_p^{s,k}$ at p. Although $V_p^{s,k} = V_q^{s,k}$, the former is of course time faulty since it arrived too late w.r.t. the sender's proper round k's broadcast event. Hence, any type of benign process failure may result in a timing failure in case of $h > 0$ and must hence be accounted for in f_s or f_a. Note that the latter is not required for the algorithm studied in this paper, since late timing failures are no more severe than omissions for the algorithm of Figure 4.
5. Although message corruption is a fairly rare event if CRC checksums are used, one has to acknowledge the fact that the occurrence of arbitrary receive link failures is more likely than it meets the eye. After all, the messages sent in *every* round of the execution may be turned into a false round k message if the round number contained in the messages is corrupted.
6. Messages with history $h > 0$ can reduce the number of arbitrary link failures, since value failures may be detected by comparison and turned into omission failures here. The number of link omission failures is

usually also decreased in case of $h > 0$. Like in the case of benign process failures, however, the number of late timing failures may increase and must be accounted for in f_ℓ^{ra} and f_ℓ^{sa}. Note that the latter is again not required for the algorithm studied in this paper, since late timing failures are no more severe than omissions for the algorithm of Figure 4.

The primary way of using our failure model in the analysis of agreement-type algorithms is the following: Given the perception vector $\mathcal{V}_q^R(\sigma_q^R)$ of some specific obedient receiver process q at its round switching time σ_q^R, it allows to determine how many perceptions will at least be present in any other obedient process r's perception vector $\mathcal{V}_r^R(\sigma_q^R + \varepsilon)$ shortly thereafter. The following Lemma 1 developed in [27] formalizes this fact.

Lemma 1 (Difference in Perceptions). *At any time t, the perception vector $\mathcal{V}_q(t)$ of any process at an obedient receiver q may contain at most $f_\ell^{ra}+f_a+f_s$ timing/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 obedient 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 $f_\ell^{ra}+f_a+f_o$ 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. All symmetric faulty perceptions present in $\mathcal{V}_q(t)$ must also be present in $\mathcal{V}_r(t+\Delta t)$, however. Summing up all the differences, the expression for Δf given in Lemma 1 follows. □

4 Clock Synchronization

Assuming that every process p 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:

(P) *Uniform Precision:* There is some constant *precision* $D_{max} > 0$ such that

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

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

(A) *Uniform Accuracy:* There are some constants R^-, O^-, R^+, $O^+ > 0$ such that

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

for any obedient active 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 (A) guarantees some relation of the progress of clock time with respect to the progress of real-time; (A) is also called *envelope requirement* in literature. Note that (P) and (A) are *uniform* [10] here, i.e., they do not only hold for non-faulty but also for benign faulty processes. Any such process must hence satisfy the clock synchronization conditions until it (possibly) crashes.

Traditional research on clock synchronization (see [22, 24,33,34] 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 [7,30,35] 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 [6, 21] are usually 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, which is roughly once per round trip. In fact, Theorem 10 will reveal that our algorithm ensures $O^- = 1/(2\tau^+)$, $O^+ = 1/(2\tau^-)$ with reasonably small R^-, R^+, where the lower bound holds only if sufficiently many processes are eventually up and running.

Note that, by using local hardware interval timers at every sender process, it is possible to arbitrarily "stretch" τ^- to obtain $\tau^+ \approx \tau^-(1+\rho)+\varepsilon$, where ε is the uncertainty resulting from the raw communication delay of the network and the execution time at the sender and receiver processes. This way, a situation comparable to the synchronous clock synchronization setting could also be established in our partially synchronous framework.

4.1 The Algorithm

The clock synchronization algorithm considered in this paper is a hybrid variant of the algorithm of [40]. It is based upon the well-known non-authenticated[11] clock synchronization algorithm of [35], which employs consistent broadcasting for generating nearly-simultaneous global resynchronization events in the system. Its pseudocode is shown in Figure 4. Note carefully that our algorithm is completely time-free in that it does not incorporate τ^+ and τ^-, and not even ε or Θ.

Comparison of the algorithm of Figure 4 with hybrid variants [16,26] of the original consistent broadcasting primitive (which do not incorporate proper system startup handling) shows that the first three **if** -clauses[12] (`lines 7-18`) are the same: Ignoring history size $h = 1$

[11] We do not consider authenticated algorithms here. Besides the disadvantages of computational and communication overhead, it is never guaranteed that malicious processes cannot break the authentication scheme. Using the algorithm of Srikanth and Toueg [35], our correctness proofs cannot be invalidated by this event.

[12] We do not count the booting-related first **if** (`line 4`) here.

```
1  VAR k : integer := 0; /* clock value */
2  VAR mode : {passive, active} := passive;

cobegin
/* Booting Section */
3  send (init, 0) to all [once]; /* join message */
4  if received (init, 0) for the first time from p
5    → re-send (init, k) and, if already sent, the last (echo, k') to p;
6  fi

/* Process for init messages */
7  if received (init, k) from at least f_ℓ^{ra} + f_a + f_s + 1 distinct processes
8    → send (echo, k) to all [once];
9  fi

/* Process for (echo, k) messages */
10 if received (echo, k) or (echo, k+1) from at least f_ℓ^{ra} + f_a + f_s + 1 distinct processes
11   → send (echo, k) to all [once];
12 fi
13 if received (echo, k) or (echo, k + 1) from at least n − f_ℓ^s − f_ℓ^r − f_a − f_s − f_o − f_c distinct processes
14   → if mode = passive → mode := active; fi
15     k := k + 1; /* update round number */
16     send (init, k) to all; /* start next round */
17 fi
/* Processes for (echo, l) messages (l ≠ k) */
18 if received (echo, l) or (echo, l + 1) from at least f_ℓ^{ra} + f_a + f_s + 1 distinct processes
19   → if l ≥ k
20     → for j := k to l − 1 send (echo, j) to all [once]; /* send skipped echos */
21       k := l; /* catch up to new clock value */
22       send (echo, k) to all;
23     fi
24 fi
25 if received (echo, l) or (echo, l + 1) from at least n − f_ℓ^s − f_ℓ^r − f_a − f_s − f_o − f_c distinct processes
26   → if mode = passive → mode := active; fi
27     if l ≥ k
28     → k := l + 1;
29       send (init, k) to all;
30     fi
31 fi
coend
```

Fig. 4. Clock Synchronization Algorithm for the Hybrid Perception based Failure Model with Startup Phase

first, each round k is started by sending an *(init, k)* message to all. When a correct process achieves sufficient evidence that at least one obedient process has sent a round k message, it sends *(echo, k)* to all (`line 7` or `line 10`). When a process can be sure that there are enough round k messages in the system, such that every obedient process will eventually achieve sufficient evidence, it advances to round $k+1$ and hence sends *(init,* $k+1$*)* (`line 13`). This guarantees both (P) and (A) if sufficiently many correct processes are up and running right from the beginning.

In reality, however, processes complete booting one after the other, at unpredictable times. Late starters could hence miss the *(echo, k)* and/or *(init, k)* message(s) of early starters. Consequently, three consecutive modes of system operation must be distinguished to properly handle system startup:

- *Early mode*, where the first few correct processes have completed booting and started exchanging messages. It terminates when the first obedient process advances its clock to 1.
- *Degraded mode*, where enough correct processes are up such that some clocks may advance when "assisted" by faulty processes and 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.

In order to add system startup handling to the original consistent broadcasting algorithm, *join* messages and two additional **if** -clauses are required. First of all, a newly booted process must tell all others that it is up now and must learn their current clock values. This is accomplished by means of *join* messages, as introduced in Section 3.2: Every process p sends *join* = *(init,* 0*)* as the very first message after having completed booting. Every process q that receives this message replies by retransmitting its previously sent *(init, k)* and *(echo, k′)* message (`line 3-6`); the latter message is omitted if q did not yet send *(echo,* 0*)* in round $k = 0$. 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 (`line 13`): There are not sufficiently many correct processes to guarantee that every correct process will eventually advance its clock when the first one does so. Here it is where the fourth **if** (`line 18`), our pivotal *catch-up rule*, comes into play: It allows a correct process to advance its clock to round l when sufficiently many *(echo, l)* or *(echo,* $l + 1$*)* messages have been received. Therefore, eventually, a sufficiently large group of correct processes can be guaranteed to be within two rounds of each other. Note, however, that the algorithm of Figure 4 cannot guarantee that the clock of an obedient process takes on every integer value: It may also leap forward due to catch-up.

The catch-up rule causes two other problems, however: First, the second and third **if**-clause (`line 10` or `line 13`) must trigger when sufficiently many *(echo)* messages from different processes within 2 rounds have been received. This is conveniently expressed in our execution model by setting the history size of *(echo, k)* messages to $h = 1$, as already mentioned: 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 $k-1$ perceptions $V_q^{p,k'}$ can safely be filled on that occasion. Note that $V_q^{p,k'}$ may be empty if q missed the proper *(echo, k)* message due to late booting.

Second, due to failures or a large group of simultaneous late joiners, the first round an initializing process might 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). A passive process switches to active mode when it has sufficient evidence that its local round number is within precision D_{max} of the clocks of the other correct active processes in the system. This is accomplished by the fifth **if**-clause (`line 25`), which triggers when sufficiently many *(echo, l)* messages have been received. We will show in Section 5 that any[13] such l can be guaranteed to be sufficiently close to the maximum correct clock value in the system.

[13] Note that there is a variant of the algorithm of Figure 4 that omits sending all the skipped *(echo, l′)* during catch-up, i.e., omits `line 20`. However, it requires a history size $h = \infty$ in the fifth **if** for ensuring that any correct processor will switch to active mode in bounded time. The alternative proof of Theorem 7 is based on a weakened version of weak relay (P3W) in Theorem 1, which ensures that *(echo,* $k' - 1$*)* for some $k' \le k$ is sent by all correct processes $\in \mathcal{P}_{up}(k-1)$ upon catch-up.

In the following sections, we will prove that our algorithm satisfies the clock synchronization properties (P) and (A). In Section 4.2, we will see how its execution matches the execution model of Section 3.1, The analysis starts with Section 5, where we explore how our algorithm behaves in early and degraded mode, i.e., when not sufficiently many processes are up to guarantee progress of the clocks. Section 6 deals with the transition between degraded and normal mode, which takes place when late processes join the system. In Section 7, we study our system in normal mode, were sufficiently many processes are participating actively in the clock synchronization algorithm.

4.2 Mapping to the Perception based Execution Model

The algorithm is given in an event based notion that consists of the 6 outermost **if** statements, numbered from the zero-th **if** (`line 4`) up to the fifth **if** (`line 25`). The variable k provides the clock value $C_p(t)$ of the processor p that executes the algorithm. The second and the third **if** are just the instances of the fourth and fifth **if** for $l = k$, which have been incorporated explicitly for the ease of explanation only and could entirely be omitted from the code. The presence of the third **if** in Figure 4 implies, however, that the only part of the fifth **if** that is ever executed is setting mode to active when in passive mode (`line 26`). Note carefully that this also happens when $l < k$. The clock value k is never changed in the fifth **if**, since k is already set to l in the fourth **if**, which always triggers before the fifth **if**.

In order to be able to analyze our algorithm under the perception based failure model of Definition 4, we must show that its execution complies to the execution model of Section 3.1. Every execution of our algorithm can in fact be described by the following processes running on every processor p:

- A single, multiple-round process K, which processes all the *(init, k)* messages in the perception vectors $\mathcal{V}_p^k$. Round switching of K is enforced externally (by other processes) by increasing the shared variable k that holds K's current round number. Only K needs to react to a *join* message of a newly booted processor, upon which it retransmits *(init, k)* and *(echo, k′)*.
- Infinitely many single-round processes L', $L' \ge 0$, where process L' processes *(echo, l)* messages for $l = L'$ in the perception vector $\mathcal{V}_p^{l'}$. All those processes must be active concurrently right from the start and can terminate after execution of `line 13` resp. `line 25`. Note that they can be viewed as a way of concurrently executing all rounds of a single multiple-round process. In active mode, any process L' with $L' < k$ can be terminated prematurely since no code remains to be executed in the fifth **if**.

In order to keep our subsequent description simple, we will usually not explicitly distinguish K and L', however: By saying that process p responds to *(init, k)* resp. *(echo, l)*, we mean that K resp. L' with $L' = l$ on processor p does so. The *(echo, k)* and *(init, k)* messages belonging to round k of the algorithm will be referred to as *tick k messages*.

In order formally model the fact that computations are actually triggered upon arrival of *(echo, x)* or *(echo, x + 1)* in our algorithm, we use messages with history size $h = 1$. For example, when an *(echo, k + 1)* message from process p arrives at q in case of $h = 1$, both the perception $V_q^{p,k+1}$ and, provided that still $V_q^{p,k} = \emptyset$, the perception $V_q^{p,k}$ is filled. Hence, the above processes can act upon a single perception vector $\mathcal{V}_q^k$ as required by our execution model, yet process messages *(echo, k)* and *(echo, k + 1)*.

5 From Early to Degraded Mode

Before considering the full system startup scenario, we will study the algorithm of Figure 4 in a system of $n \geq f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ processes,[14] where a fixed but arbitrary number $0 \leq n_{up}(K) \leq n$ of processes have completed booting when the messages from the first obedient process that started some round K may arrive. Note that all those $n_{up}(K)$ processes can receive all messages for ticks $k \geq K$ from each other (except faulty ones, of course). The special case $n_{up}(0) = n$ describes the situation where all processes must be up and running right from the very beginning, i.e., where system startup is ignored [26, 35]. The case $n_{up}(1) < n$ will be used to describe the behavior of our algorithm during (the beginning of) degraded mode.

We start with some frequently used definitions and lemmas, which apply to any operation mode. Note that a passive process p's local clock $C_p(t)$ refers to its current round number k (see Figure 4) at time t in our analysis; terms like *p's clock value* and *"p sets its clock"* are hence meaningful for both active and passive processes. Still, the clock synchronization properties need only hold for active processes. Of particular importance is the maximum clock value of all obedient processes in the system, which will be used as the "reference time" in our precision analysis.

Definition 5 (Maximum Clock Value). *Consider an arbitrary real-time t.*

- $C_{\max}(t)$ *is the maximum of the local clocks of all obedient processes that are up at time t.*
- $\sigma_{first}^{k-1} = \sigma_p^{k-1} \leq t$ *denotes the real-time when the first obedient process p sets its local clock to* $k = C_{\max}(t)$*;* σ_{first}^{-1} *is the time when the first obedient process has completed booting. Note that p's clock value, and hence* $C_{\max}$*, is defined to be k at time* σ_{first}^{k-1}*, and* $k-1$ *at* $\sigma_{first}^{k-1} - \epsilon$ *for any infinitesimally small* $\epsilon > 0$.

The maximum set of obedient processes that did not miss any tick k message from each other (except the ones missing due to process or link failures, of course) is denoted by $\mathcal{P}_{up}(k)$*, with* $n_{up}(k) = |\mathcal{P}_{up}(k)|$.

The following Lemma 2 reveals a few straightforward properties about messages sent by correct[15] processes.

Lemma 2 (Message Generation). *Consider any correct process p executing the algorithm of Figure 4.*

(1) If p reached clock value $k > 0$ by time t, it must have sent (echo, k') for all $0 \leq k' \leq k-1$ by time t.
(2) If p reached clock value $k > 0$ via the third **if** *(`line 13`) at time t, then (init, k) has also been emitted by time t.*
(3) If p reached clock value $k > 0$ via the catch-up rule (`line 18`) at time t, then its (echo, k) has also been emitted by time t.

Proof. Consider the sequence of values $\{c_m\}_{m \geq 0}$, assigned to p's clock variable at real-times $\{t_m\}_{m \geq 0}$. Using induction on m, we will show that p must have emitted *(echo, l')* for all $0 \leq l' \leq c_m - 1$ (resp. $l' \leq c_m$ in case of using the catch-up rule) by time t_m. Since it is obvious from Figure 4 that c_m can only be monotonically increasing, our lemma then follows immediately.
For the basis $m = 0$, obviously $c_0 = 0$ and the claim is void since no *echo*-message needs to have been sent by time t_0. Assuming now that the claim holds for $m-1 \geq 0$, i.e., all *(echo, l')* for $0 \leq l' \leq c_{m-1} - 1$ have been sent by time t_{m-1} when p's clock is set to c_{m-1}, we have to show that the claim holds for m as well: Three cases must be distinguished here, according to whether setting p's clock to c_m occurs via the third, fourth or fifth **if**.
If p sets its clock via the third **if** in in Figure 4, we must obviously have $c_m = c_{m-1} + 1$. Since triggering of the third **if** implies triggering of the second **if**, *(echo, $c_m - 1$) = (echo, c_{m-1})* must have been sent by time t_m. Combining this with the induction hypothesis, the claim follows immediately. Sending *(init, c_m)* as asserted in item (2) is obvious from the code.
If p sets its clock to c_m via the fourth **if**, it sends *(echo, l'')* for $c_{m-1} \leq l'' \leq c_m$ according to `line 20`; note that *(echo, c_{m-1})* is of course only emitted here if this has not happened earlier. Combining this with the induction hypothesis, our claim—as well as item (3)—follows also in this case. Finally, we mentioned already in our description of the algorithm that p cannot set its clock via the fifth **if**, since the fourth **if** triggers before and sets $k := l$, such that the third **if** takes over here. □

The following Lemma 3 shows that progress of $C_{\max}(t)$ is only possible via the third **if** (`line 13`) in Figure 4.

Lemma 3 (3rd if). *The first obedient process that sets its clock to tick $k = C_{\max}(t) > 0$ by time t must do so by the third* **if** *(`line 13`) and hence, if correct, sends (echo, $k-1$) and (init, k) by time t.*

[14] Incorporating f_ℓ^s in our algorithm is only required to guarante liveness in the transition from degraded to normal mode (Theorem 5 and Theorem 7); it accounts for send link failures in *join*-messages of late starters. Consequently, most of our results—in particular, the ones of this section and all precision-related ones—hold when setting $f_\ell^s = 0$ in Figure 4 and in our formulas (despite of $f_\ell^r > 0$) as well.

[15] Transmission of messages of an obedient process q can be guaranteed to be successful only if q is correct.

Proof. By contradiction. Assume that the first obedient process p sets its clock to $k = C_{\max}(t)$ at instant t by the catch-up rule (`line 18`) [or the fifth **if** (`line 25`)]. At least one obedient process must have sent a message for a tick $l \geq k$ to enable those rules at p. Since obedient processes only send messages for ticks less or equal their local clock value, at least one must already have had a clock value $l \geq k$ at instant t. This contradicts p to be the first one that reaches clock value k. Sending *(echo, $k-1$)* and *(init, k)* follows from Lemma 2. □

The following Lemma 4 shows that processes that completed booting early enough can be guaranteed to be in $\mathcal{P}_{up}(k)$, since they must have got all tick k messages from other obedient processes (except messages hit by failures). Bear in mind, however, that early booting is not the only only way of ensuring inclusion in $\mathcal{P}_{up}(k)$. A late starting process p may also achieve $p \in \mathcal{P}_{up}(k)$ via the *join*-protocol even though it completes booting after time $\sigma_{first}^{k-1} + \tau^-$.

Lemma 4 (Booting Times). *Any obedient process p that is up by time $\sigma_{first}^{k-1} + \tau^-$ satisfies $p \in \mathcal{P}_{up}(k)$.*

Proof. Since the first obedient process that emits a tick k message does so not earlier than time σ_{first}^{k-1}, according to our algorithm in Figure 4 and Lemma 3, no obedient process p can receive such a message before time $\sigma_{first}^{k-1} + \tau^-$. If p is up by that time, it will hence receive all tick k messages from correct processes. □

The following Theorem 1 is the first major result of this section. It shows that the clocks of obedient processes obey two properties, namely, unforgeability (P2), and weak relay (P3W), the names of which have been coined in the context of consistent broadcasting [15, 35, 36]. Our properties are weak ones, since we do not have sufficiently many processes up and running to ensure the strong ones of Theorem 6 which also guarantee progress.

Theorem 1 (Weak Clock Synchronization Properties). *For $n \geq f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$, the algorithm from Figure 4 achieves*

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

P3W Uniform Weak Relay. For any $k \geq 1$, if an obedient process sets its clock to k at time t, then, by time $t+\varepsilon$, every obedient process $\in \mathcal{P}_{up}(k-1)$ sets its clock at least to $k-1$ and, if correct[16], sends (echo, $k-1$).

Proof. Uniform Unforgeability. Setting the clock value can be done by (1) the third **if** (`line 13`) or (2) the fourth **if** (`line 18`). The proofs are by contradiction.
Assume (1) that there is an obedient process p that sets its clock to $k+1$ before instant $t+2\tau^-$ using the third **if**. This implies $|\mathcal{V}_p^{k'}(t+2\tau^-)| \geq 2f_\ell^{ra} + f_\ell^r + 2f_a + f_o + f_s + 1$. Since, according to Lemma 1, only at most $f_\ell^{ra} + f_a + f_s$ of the corresponding *(echo, k)* perceptions may be due to messages produced by arbitrary receive link failures and time/value-faulty processes, at least one obedient process q must have sent *(echo, $\overline{k}$)* for $\overline{k} \in \{k, k+1\}$ before time $t + \tau^-$. Process q has done so caused by received messages for the $\overline{k}$-th tick such that $|\mathcal{V}_q^{\overline{k}}(t + \tau^-)| \geq f_\ell^{ra} + f_a + f_s + 1$ or $|\mathcal{V}_q^{\overline{k}'}(t+\tau^-)| \geq f_\ell^{ra} + f_a + f_s + 1$. Hence—by the same argument as before—at least one obedient process must have sent a message for the $\overline{k}$-th tick before time t, which contradicts the assumption of unforgeability.
Assume (2) that there is an obedient process p that sets its clock to $k+1$ before instant $t + 2\tau^-$ using the fourth **if**. Process p does so because $|\mathcal{V}_p^{l'}(t + 2\tau^-)| \geq f_\ell^{ra} + f_a + f_s + 1$ for some $l \geq k+1$. That is, at least one *(echo, $\overline{l}$)* message, $\overline{l} \in \{l, l+1\}$, must have been sent by an obedient process q before $t + \tau^-$. Process q has sent it, because $|\mathcal{V}_q^{\overline{l}}(t + \tau^-)| \geq f_\ell^{ra} + f_a + f_s + 1$ or $|\mathcal{V}_q^{\overline{l}'}(t + \tau^-)| \geq f_\ell^{ra} + f_a + f_s + 1$. The latter perceptions stem from messages that must have been sent before time t, since $\overline{l} > k$ and messages for tick $\overline{l}$ are sent by an obedient process only after tick k messages. But by assumption (P2), no tick k message was sent before t, which again provides the required contradiction.

Uniform Weak Relay. Assume first that $k = C_{\max}(t)$. If the obedient process p that advanced its clock to k by assumption used the third **if**, then $|\mathcal{V}_p^{(k-1)'}(t)| \geq 2f_\ell^{ra} + f_\ell^r + 2f_a + f_o + f_s + 1$. Hence, the perception vector for tick $k-1$ at any obedient process $q \in \mathcal{P}_{up}(k-1)$ 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 1.[17] It follows that every obedient process $\in \mathcal{P}_{up}(k-1)$ achieves sufficient evidence in the fourth **if** to catch-up to round $k-1$ (if it has not already done so). This fact (or Lemma 2 if p has already reached an even larger clock value $\geq k$ by $t + \varepsilon$) guarantees that *(echo, $k-1$)* is sent by time $t + \varepsilon$.
If, on the other hand, the obedient process p advanced its clock to k by the fourth **if**, there must be a process p' that set its clock to k by the third **if** earlier according to Lemma 3. Using p' in the above argument establishes (P3W) in this case as well.
If $k < C_{\max}(t)$, then at least one obedient process has already set its clock to $l > k$ at some time $t' \leq t$ using the third **if**. We have shown in the previous paragraph that all obedient processes $\in \mathcal{P}_{up}(k-1)$ must set their clocks to $l-1$ (and hence at least to $k-1$) by time $t' + \varepsilon$ (and hence by time $t + \varepsilon$) as asserted. □

The following simple Lemma 5, which will be used frequently in our proofs, follows immediately from prop-

[16] Recall that the transmission of an obedient process can be guaranteed to be successful only if this process is correct.

[17] If n is increased, $|\mathcal{V}_q^{(k-1)'}(t + \varepsilon)|$ increases accordingly. This fact will be used for ensuring weak relay (P3W) despite of some missing *(echo, $k-1$)* due to send link failures: By adding another f_ℓ^s to n, the at most f_ℓ^s lost *join*-messages of a late starter p, which may lead to the lack of up to f_ℓ^s responses *(echo, $k-1$)* at p, can be masked transparently, see the proof of Theorem 5.

erty (P2). It ensures that $\sigma^k_{first} - \sigma^{k-1}_{first} \geq 2\tau^-$ for any $k \geq 0$.

Lemma 5 (Fastest Progress). *Let p be the first obedient process that sets its clock to k at time t. Then no obedient process can reach a larger clock value k' before $t + 2\tau^-(k' - k)$.*

Proof. By induction on $l = k' - k$. For $l = 1$, Lemma 5 is identical to unforgeability and therefore true. Assume that no obedient process has set its clock to $k + l$ before $t+2\tau^- l$ for some l. Thus no obedient process may set its clock to $k+l+1$ before $t+2\tau^- l+2\tau^- = t+2\tau^-(l+1)$ by unforgeability. Hence Lemma 5 is true for $l + 1$ as well. □

In our analysis, we will frequently require a bound upon the maximum progress of $C_{\max}$ during a given real-time interval $[t_1, t_2]$. Lemma 5 can be applied for this purpose if $t_1 = \sigma^k_{first}$, but not if t_1 is arbitrary. As in [28], we unify those cases via the following Definition 6.

Definition 6 (Synchrony). *Real-time t is in synchrony with $C_{\max}(t)$ iff $t = \sigma^k_{first}$ for some arbitrary $k \geq -1$. Let the indicator function of non-synchrony be defined as*

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

Lemma 6 (Maximum Increase of $C_{\max}$ within Time Interval). *For any two real-times $t_2 \geq t_1$,*

$$C_{\max}(t_2) - C_{\max}(t_1) \leq \left\lfloor \frac{t_2 - t_1}{2\tau^-} \right\rfloor + I_\sigma(t_1).$$

Proof. Let $k = C_{\max}(t_1) - 1$. We have to distinguish the two cases $\sigma^k_{first} = t_1$ and $\sigma^k_{first} < t_1$.
Let $\sigma^k_{first} = t_1$ such that $I_\sigma(t_1) = 0$. From Lemma 5, it 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 6 is true in this case.
Now let $\sigma^k_{first} < t_1$ such that $I_\sigma(t_1) = 1$ and abbreviate $t' = \sigma^{k+1}_{first} > t_1$. We can now apply Lemma 5 starting from time t'. Since $t_2 - t_1 > t_2 - t'$ it follows from Lemma 5 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', $C_{\max}$ increases by one 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 6 is also true in this case. □

The following two technical Lemmas 7 and 8 are the major tools in our precision analysis. Both lemmas—and hence all the results built upon those—actually hold for every algorithm that satisfies uniform unforgeability (P2) and uniform weak relay (P3W). The first Lemma 8 bounds the maximum real-time difference between any obedient process's clock and $C_{\max}$ (i.e., the fastest obedient process) at about the same clock time. Note carefully that it actually holds for any $k' \geq k$ and hence for any $t \geq \sigma^k_{first}$, since $p \in \mathcal{P}_{up}(k)$ implies $p \in \mathcal{P}_{up}(k')$.

Lemma 7 (Real-Time Bounds). *For any obedient process $p \in \mathcal{P}_{up}(k)$ where $k+1 = C_{\max}(t)$ such that $\sigma^k_{first} \leq t < \sigma^{k+1}_{first}$, the following holds:*

$$\sigma^{k-1}_p \leq \sigma^k_{first} + \varepsilon \tag{5}$$

$$\begin{aligned} C_p(t+\varepsilon) &\geq C_p(\sigma^{k-1}_p) \\ &= C_{\max}(\sigma^k_{first}) - 1 = C_{\max}(t) - 1 \end{aligned} \tag{6}$$

Moreover, if p is correct, it is guaranteed to send (echo, k) by time $t + \varepsilon$.

Proof. According to weak relay (P3W) in Theorem 1, all obedient processes $\in \mathcal{P}_{up}(k)$, and hence also p, must set their clocks to at least k by time $\sigma^k_{first} + \varepsilon$. (P3W) hence implies (5) and the claimed sending of *(echo, k)*.
If (5) holds, it follows from $\sigma^k_{first} \leq t < \sigma^{k+1}_{first}$ that $C_{\max}(t) = k + 1$ and $\sigma^{k-1}_p \leq \sigma^k_{first} + \varepsilon \leq t + \varepsilon$. Consequently, $C_p(t+\varepsilon) \geq C_p(\sigma^{k-1}_p) = k$ and hence $C_{\max}(t) - C_p(t + \varepsilon) \leq 1$, which confirms (6). □

The second Lemma 8 provides a bound for the maximum clock time difference of any obedient process and $C_{\max}$ at the same real-time. Note that the precision D_{MCB} implied by Lemma 8 actually holds for any $k' \geq k$ and hence for any $t \geq \sigma^{k-2}_p$, since $p \in \mathcal{P}_{up}(k)$ implies $p \in \mathcal{P}_{up}(k')$.

Lemma 8 (Logical Time Bounds). *For any obedient process $p \in \mathcal{P}_{up}(k)$, $k \geq 1$, and any t' with $\sigma^{k-2}_p \leq t' < \sigma^{k-1}_p$,*

$$C_{\max}(t') \leq C_p(t') + D_{MCB} = k - 1 + D_{MCB} \tag{7}$$

$$\begin{aligned} C_{\max}(\sigma^{k-1}_p) &\leq C_p(\sigma^{k-1}_p) + D_{MCB} - 1 \\ &= k - 1 + D_{MCB} \end{aligned} \tag{8}$$

with

$$D_{MCB} = \left\lfloor \frac{1}{2}\Theta + \frac{3}{2} \right\rfloor. \tag{9}$$

Proof. If $C_{\max}$ never reaches $k + 1$, (7) and (8) hold trivially since $D_{MCB} \geq 1$. Hence, suppose that $C_{\max}$ reaches $k + 1$ at time $\sigma^k_{first} < \infty$. Let $t = \sigma^{k-1}_p$ denote the time when p sets its clock to k. In order to compute $C_{\max}(t)$, we use Lemma 6 and our knowledge of $C_{\max}(t - \varepsilon) \leq C_{\max}(\sigma^k_{first}) = k+1$ due to monotonicity of $C_{\max}$ in conjunction with (5): From Lemma 6, we infer that $C_{\max}(t) \leq \lfloor \frac{t-(t-\varepsilon)}{2\tau^-} \rfloor + I_\sigma(t-\varepsilon) + C_{\max}(t-\varepsilon)$. If $\sigma^k_{first} = t - \varepsilon$ such that $t - \varepsilon$ is synchronized with $C_{\max}$, then $C_{\max}(t - \varepsilon) = k + 1$ and $I_\sigma(t - \varepsilon) = 0$ according to Definition 6. If, on the other hand, $\sigma^k_{first} > t - \varepsilon$, then $C_{\max}(t - \varepsilon) = k$ and $I_\sigma(t - \varepsilon) = 1$. In both cases, $I_\sigma(t - \varepsilon) + C_{\max}(t - \varepsilon) = k + 1$ and hence $C_{\max}(t) \leq \lfloor \frac{\varepsilon}{2\tau^-} \rfloor + k + 1 \leq \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor + k - 1$. As $C_{\max}(t') \leq C_{\max}(t)$ since $t' < t$ and $C_p(t') = k - 1$, (7) follows. If we increase t' until $t' = t = \sigma^{k-1}_p$, then $C_p(t')$ increases from $k - 1$ to k and thus reduces the difference to $C_{\max}$ by 1, as asserted in (8). □

The following Lemma 9 is a simple corollary of Lemma 8. It immediately leads to the precision bound given in Theorem 2 below.

Lemma 9 (Precision Bound). *Let t be any real-time where $C_{\max}(t) = K \geq 1$, and p be an obedient process with $C_p(t) = k$ for some $0 \leq k \leq K$. If $p \in \mathcal{P}_{up}(k+1)$, then $k \geq K - D_{MCB}$. If $t = \sigma_p^{k-1}$ and $p \in \mathcal{P}_{up}(k)$, then $k \geq K - D_{MCB} + 1$.*

Proof. Inequality (7) in Lemma 8 reveals that $C_{\max}(t') \leq k + D_{MCB}$ for all $\sigma_p^{k-1} \leq t' < \sigma_p^k$ since $p \in \mathcal{P}_{up}(k+1)$ by assumption. Given that $C_p(t) = k$ implies $t < \sigma_p^k$ here, we can choose $t' = t$ such that $K = C_{\max}(t) \leq k + D_{MCB}$ as asserted. If, on the other hand, $t = \sigma_p^{k-1}$, then (8) applies and hence $K = C_{\max}(t) \leq k + D_{MCB} - 1$. □

Theorem 2 (Bulk Precision). *Consider a system of $n \geq f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ processes running the algorithm of Figure 4. For any $K \geq 1$ and all real-times $t \geq \sigma_{first}^{K-1}$, all obedient processes $p \in \mathcal{P}_{up}(k_p + 1)$ with $k_p = C_p(\sigma_{first}^{K-1})$ maintain a mutual precision of $D_{MCB} = \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor$.*

Proof. Let $K' \geq K$ be such that $\sigma_{first}^{K'-1} \leq t < \sigma_{first}^{K'}$ and hence $C_{\max}(t) = K'$. Applying Lemma 9 reveals that all obedient processes $p \in \mathcal{P}_{up}(k'_p + 1)$ with $k'_p = C_p(t)$ satisfy $K' - k'_p \leq D_{MCB}$ and are hence within precision D_{MCB}. Since $p \in \mathcal{P}_{up}(k_p + 1)$ implies $p \in \mathcal{P}_{up}(k'_p + 1)$ for any $k'_p \geq k_p$, our lemma follows. □

With those preparations, we are now ready for computing the precision of our algorithm during early and (the beginning of) degraded mode: Since no obedient process can advance its clock from 0 to 1 as long as less than $n - f_\ell^s - f_\ell^r - f_a - f_s - f_o - f_c$ processes have completed booting, see Figure 4 (`line 13 and 25`), the system remains in early mode until the $n - f_\ell^s - f_\ell^r - f_a - f_s - f_o - f_c$-th process p (faulty ones included) completes booting, at some time t_{up}. Progress of any obedient process—and hence the transition to degraded mode—is possible only after time $t_{up} + \tau^-$, since p's *(echo, 0)*, as well as the assistance of all faulty processes, is needed for this purpose. Hence, degraded mode starts at some time $\sigma_{first}^0 \geq t_{up} + \tau^-$, with some specific number $n - f_\ell^s - f_\ell^r - f_a - f_s - f_o - f_c \leq n_{up}(1) \leq n$ of processes that have completed booting soon enough to be able to receive each other's tick 1 messages. Note that all these $n_{up}(1)$ processes have clock value 0 or 1, and are hence a priori synchronized to each other within precision 1 during early mode.

Setting $K = 1$ in Theorem 2 immediately yields the precision of our algorithm in a system with $n \geq f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ processes, where an arbitrary but fixed number $n_{up}(1) \leq n$ of processes start in degraded mode. It shows that all obedient processes among the $n_{up}(1)$ ones are synchronized to within D_{MCB} for all times.

Theorem 3 (Precision of Early Starters). *Consider a system of $n \geq f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 2f_s + 2f_o + f_c + 1$ processes with a fixed number $n_{up}(1) = |\mathcal{P}_{up}(1)| \leq n$ of processes that completed booting by time σ_{first}^0 and are hence ready to receive all their tick 1 messages. The obedient processes $p \in \mathcal{P}_{up}(1)$ are synchronized to each other within $D_{MCB} = \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor$ during whole system lifetime.*

Proof. Since all obedient processes in $\mathcal{P}_{up}(1)$ have clock value 0 or 1, they are obviously synchronized within precision $1 \leq D_{MCB}$ up to time σ_{first}^0. From time σ_{first}^0 on, Theorem 2 secures precision D_{MCB} due to the fact that all obedient processes p with $C_p(\sigma_{first}^0) = k \leq 1$ are up since time $\sigma_{first}^0 \leq \sigma_{first}^k$ according to our assumption $p \in \mathcal{P}_{up}(1)$. □

6 From Degraded to Normal Mode

Theorem 3 can be applied to all obedient processes that have completed booting by the beginning of degraded mode. It shows that all those early starters remain synchronized to each other within D_{MCB}. The problem is, however, that additional processes might complete booting during degraded mode, after the set of early starters has made some progress. As late processes start with a clock value of 0, they are clearly not synchronized right from the start. This does not matter, since passive processes need not satisfy precision (P). Unfortunately, however, even active late starters could compute their first clock values from very little information on the system state. Although they will eventually also reach precision D_{MCB}, this results in a considerably larger precision D_{max} during a short period of time after becoming active. Since no process can determine locally whether it is already within D_{MCB}, it is D_{max} that determines our algorithm's worst case precision.

Besides the larger precision D_{max}, late starters introduce another disadvantageous property: In order to guarantee that all correct processes become active and synchronized within bounded time, we have to guarantee a sufficiently large "leader group" of correct processes. Therefore, our algorithm requires an increased number of processes for some restricted failure modes and for tolerating send link failures[18] in *join*-messages: It will turn out that $f_s + f_c + f_\ell^s$ additional processors are required here, i.e., that we need a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes. Unlike in Section 5, symmetric process failures are hence as severe as asymmetric ones, and crash failures are as severe as our omission failures in presence of late starters. Note that this can be explained by the fact that (receiver) processes boot at unpredictable times, which turns otherwise consistently perceived process failures into inconsistently perceived ones.

[18] Send link failures ($f_\ell^s > 0$) must be considered explicitly only in Theorem 5 and Theorem 7; all other results hold for $f_\ell^s = 0$ (despite of $f_\ell^r > 0$) as well.

In order to compute D_{max}, we first give a bound on the real-time t_{sync} when an obedient late starter process p reaches a local clock value $C_p(t_{sync})$ within precision D_{MCB}, such that $C_{\max}(t) - C_p(t) \le D_{MCB}$ for any time $t \ge t_{sync}$. Time t_{sync} hence delimits the period where p may have the worst case precision D_{max}. It must be noted, however, that t_{sync} depends upon progress of $C_{\max}$ here.

Lemma 10 (First Synchronization). *Let p be an obedient process that gets up at time t_{up} and let $k = C_{\max}(t_{up} - \tau^- - \epsilon) + 1 \ge 1$ for any infinitesimally small ϵ. Then, $p \in \mathcal{P}_{up}(k)$, $C_p(t_{sync}) = k$ where $t_{sync} \le \sigma^k_{first} + \varepsilon$, and*

$$C_{\max}(t) - C_p(t) \le D_{MCB} \tag{10}$$

for all times $t \ge t_{sync} = \sigma^{k-1}_p$. Moreover,

$$C_{\max}(t_{sync} - \epsilon) \le C_{\max}(t_{up} - \tau^- - \epsilon) + \left\lfloor \frac{1}{2}\Theta + \frac{3}{2} \right\rfloor. \tag{11}$$

Proof. Our choice of k ensures $\sigma^{k-1}_{first} \ge t_{up} - \tau^-$ and hence $t_{up} \le \sigma^{k-1}_{first} + \tau^-$, such that $p \in \mathcal{P}_{up}(k)$ by Lemma 4. Hence, Lemma 7 holds and (5) ensures $t_{sync} = \sigma^{k-1}_p \le \sigma^k_{first} + \varepsilon$ and hence $C_p(t) \ge k$ for all $t \ge t_{sync}$. Lemma 9 guarantees that p is within precision D_{MCB} for all $t \ge t_{sync}$ as asserted in (10).
In oder to confirm (11), we abbreviate $t' = \sigma^k_{first}$ such that $C_{\max}(t') = C_{\max}(t_{up} - \tau^- - \epsilon) + 2 = k + 1$ and $I_\sigma(t') = 0$ according to Definition 6. From Lemma 6, it follows that $C_{\max}(t_{sync} - \epsilon) \le C_{\max}(t' + \varepsilon - \epsilon) \le C_{\max}(t') + \lfloor \frac{(t'+\varepsilon-\epsilon)-t'}{2\tau^-} \rfloor + I_\sigma(t') \le C_{\max}(t_{up} - \tau^- - \epsilon) + 2 + \lfloor \frac{1}{2}\Theta - \frac{1}{2} \rfloor = C_{\max}(t_{up} - \tau^- - \epsilon) + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor$ as asserted. □

Next we have to consider the worst case quality of the information that a passive process p might use to change to active. We start with Lemma 11, which reveals that messages from some common obedient process r are required for both advancing $C_{\max}$ (`line 13`) and changing p to active (`line 25`). Note carefully, however, that it need not be the same message from r since $h = 1$.

Lemma 11 (Common Process). *For $n \ge 2f^s_\ell + 2f^{ra}_\ell + 2f^r_\ell + 3f_a + 3f_s + 2f_o + 2f_c + 1$, any two obedient processes p and q that execute either the third or the fifth* **if** *(*`line 13` *or* `line 25`*) in the algorithm of Figure 4 use messages from at least one common obedient process r.*

Proof. Let $f'_a \le f_a$ and $f'_s \le f_s$ be the actual number of arbitrary and symmetric faulty processes, respectively. Both processes p and q must have got at least $n - f^s_\ell - f^r_\ell - f_a - f_s - f_o - f_c$ *(echo, k_p)* and *(echo, k_q)* perceptions, respectively. At most $f'_a + f'_s + f^{ra}_\ell$ of those could have a malign faulty origin (see Lemma 1), such that $m = n - f^s_\ell - f^r_\ell - f_a - f_s - f_o - f_c - f'_a - f'_s - f^{ra}_\ell$ is the minimal number of perceptions obtained from obedient processes at either p and q. The total number of obedient processes in the system is at most $n - f'_a - f'_s$. Since

$$\begin{aligned} 2m - n + f'_a + f'_s &= n - 2f^s_\ell - 2f^r_\ell - 2f_a - 2f_s - 2f_o \\ &\quad -2f_c - f'_a - f'_s - 2f^{ra}_\ell \\ &\ge n - 2f^s_\ell - 2f^r_\ell - 2f^{ra}_\ell - 3f_a - 3f_s \\ &\quad -2f_o - 2f_c \\ &\ge 1, \end{aligned} \tag{12}$$

the pigeonhole principle reveals that at least one of p and q's *echo*-perceptions must originate from the same obedient process r (although it need not be the same message from r that generated those perceptions at p and q). □

We are now ready for computing D_{max}. The worst situation with respect to $C_{\max}$ occurs when a late starter p switches to active shortly after getting up, such that is has very few information on the system state, and that $C_{\max}$ progresses at maximum rate until time t_{sync}, where p is guaranteed to be within D_{MCB}. The worst case precision D_{max} then occurs shortly before t_{sync}.

Theorem 4 (Precision). *For $n \ge 2f^s_\ell + 2f^{ra}_\ell + 2f^r_\ell + 3f_a + 3f_s + 2f_o + 2f_c + 1$, the algorithm of Figure 4 achieves the precision property (P) during the whole system lifetime with $D_{max} = \lfloor \Theta + 2 \rfloor$.*

Proof. Let t_{up} be the time when the late starting process p gets up. Assume that process p changes to active at time $t_1 \ge t_{up}$, with a clock value of $k+1$, based on an *(echo, $k+1$)* message received at time $t'_1 \le t_1$ that was sent by the common obedient process r guaranteed by Lemma 11. This *(echo, $k+1$)* message is "interpreted" as *(echo, k)* at p, which is legitimate due to the history size $h = 1$. In addition, consider time $t_2 = \sigma^{k+1}_{first}$, when the first obedient process q increases $C_{\max}$ to $k+2$. According to Lemma 11, an *(echo, $k+1$)* message from the same process r must be present at q at time $t'_2 \le t_2$. We must distinguish two cases here: The *(echo, $k+1$)* message received at p could be the result of r's "regular" *(echo, $k+1$)* broadcast, or r's response to p's *join*-message.
If *(echo, $k+1$)* at p and q are from r's broadcast, $|t'_1 - t'_2| \le \varepsilon$. The worst case precision occurs when r sent *(echo, $k+1$)* as early as possible, such that $C_{\max}$ can progress as much as possible before p gets the associated messages (reception of these messages would require p to catch up to a clock value $> k+1$). The worst setting is hence $t'_1 = t_{up}$ and $t_2 = t'_2 = \sigma^{k+1}_{first} = t_{up} - \varepsilon$. Since $t_2 = \sigma^{k+1}_{first}$, t_2 is synchronized with $C_{\max}$ and hence $I_\sigma(t_2) = 0$ by Definition 6. Lemma 6 hence reveals

$$\begin{aligned} C_{\max}(t_{up} - \tau^- - \epsilon) &\le C_{\max}(t_2) + \\ &\quad \left\lfloor \frac{(t_{up} - \tau^- - \epsilon) - (t_2)}{2\tau^-} \right\rfloor + I_\sigma(t_2) \\ &= k + 2 + \left\lfloor \frac{\varepsilon - \tau^- - \epsilon}{2\tau^-} \right\rfloor \\ &\le \left\lfloor \frac{1}{2}\Theta \right\rfloor + k + 1 \end{aligned} \tag{13}$$

for any infinitesimally small ϵ here.
If, alternatively, p gets its *(echo, $k+1$)* as a response to its *join*-message, process r must not have catched up to

some clock value $> k+1$. Hence, if t_0 denotes the real-time when *join* arrives at r, we must have $t_0 < \sigma_r^{k+1} \leq \sigma_{first}^{k+2} + \varepsilon$ (and hence $t_1' < \sigma_{first}^{k+2} + \varepsilon + \tau^+$ since the response message may travel at most τ^+). After all, $r \in \mathcal{P}_{up}(k+2)$ since r helped to advance $C_{\max}$ to $k+2$, such that (5) in Lemma 7 implies $\sigma_r^{k+1} \leq \sigma_{first}^{k+2} + \varepsilon$. Allowing *join* to travel only τ^- in order to maximize t_{up} in our worst case setting, this implies $t_{up} < \sigma_{first}^{k+2} + \varepsilon - \tau^-$. Abbreviating $t_3 = \sigma_{first}^{k+2}$, this implies

$$
\begin{aligned}
C_{\max}(t_{up} - \tau^- - \epsilon) &\leq C_{\max}(t_3) + \lfloor \frac{\varepsilon - 2\tau^- - \epsilon}{2\tau^-} \rfloor \\
&= k + 3 + \lfloor \frac{\tau^+ - 3\tau^- - \epsilon}{2\tau^-} \rfloor \\
&\leq \lfloor \frac{1}{2}\Theta + \frac{1}{2} \rfloor + k + 1 \qquad (14)
\end{aligned}
$$

here. Comparison with (13) reveals that this constitutes the worst case situation.

We are still free to choose t_1 to complete our worst case setting for D_{max}. Note that p could switch to active based on messages with malign origin or sent by passive processes, which means that $t_1 \geq t_1'$ can be arbitrary. We assume that this happens immediately before t_{sync}, the time when p is guaranteed to be within precision D_{MCB} according to Lemma 10. Recalling (11), the required bound for D_{max} is hence implied by $C_{\max}(t_{sync} - \epsilon) \leq C_{\max}(t_{up} - \tau^- - \epsilon) + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor \leq \lfloor \frac{1}{2}\Theta + \frac{1}{2} \rfloor + k + 1 + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor \leq \lfloor \Theta + 2 \rfloor + k + 1$, such that $C_p(t_1) = k + 1 \geq C_{\max}(t_{sync} - \epsilon) - \lfloor \Theta + 2 \rfloor$ as asserted. □

Apart from precision, it is not difficult to show that all active processes also satisfy the upper envelope bound of the accuracy property (A):

Lemma 12 (Upper Envelope Bound). *For $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$, all obedient processes p that are active by time t_1 satisfy $C_p(t_2) - C_p(t_1) < \frac{t_2 - t_1}{2\tau^-} + D_{max} + 1$ for all times $t_2 \geq t_1$.*

Proof. From Theorem 4, it follows that $C_{\max}(t) - D_{max} \leq C_p(t) \leq C_{\max}(t)$ at all times $t \geq t_1$. Thus, $C_p(t_2) - C_p(t_1) \leq C_{\max}(t_2) - C_{\max}(t_1) + D_{max}$. Applying Lemma 6, the statement of our lemma follows immediately. □

Theorem 4 reveals that late starters can be guaranteed to maintain precision when they switch to active mode. What still needs to be shown, however, is that this switch to active occurs eventually. In order to prove this, we first show in Lemma 13 that, at any time t, at least $f_\ell^s + f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes have clock values of $C_{\max}(t)$ or $C_{\max}(t) - 1$ and emitted corresponding *echo*-messages by time t.

Lemma 13 (Frontier Processes). *Consider any time t with $k = C_{\max}(t) > 0$. In a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes, the algorithm of Figure 4 ensures that there are at least $f_\ell^s + f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes with local clock values k or $k-1$ which have sent (echo, k) or (echo, $k-1$).*

Proof. Let p be the first obedient process that sets its clock to the maximum at time t, such that $C_p(t) = C_{\max}(t) = k$. According to Lemma 3, this happens via the third **if** (`line 13`), so p satisfies $|\mathcal{V}_p^{(k-1)'}(t)| \geq n - f_\ell^s - f_\ell^r - f_a - f_s - f_o - f_c \geq f_\ell^s + 2f_\ell^{ra} + f_\ell^r + 2f_a + 2f_s + f_o + f_c + 1$. At least $f_\ell^s + f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ of those originate from correct processes that sent either *(echo, $k-1$)* or *(echo, k)*, recall the history size $h = 1$. Since correct processes never send echo-messages for ticks larger than their local clock values, they must all have a clock value of k or $k-1$ at time t. □

The following major Theorem 5 implies a bound on the time t_{sync} when a late starting process p gets synchronized within D_{MCB}, which—unlike Lemma 10—does not depend upon progress of $C_{\max}$. The problem with Lemma 10 is that one cannot a priori bound the difference between t_{up} and $t_{sync} \leq \sigma_{first}^k + \varepsilon$, since σ_{first}^k can happen at any time in the future. In this case, however, p actually gets synchronized when the *join*-protocol terminates, i.e., by time $t_{up} + 2\tau^+$: Since there is no further progress of $C_{\max}$ here, the responses to *join* recover all missed tick $k-1$ or $k-2$ messages. Having received all the responses, the late starter p can hence catch up to $k-2$ as if it had been up right from time σ_{first}^{k-3} on, i.e., as if $p \in \mathcal{P}_{up}(k-2)$. A similar argument applies if $C_{\max}$ progresses to k during the *join*-protocol. It is important to note, however, that p need not be active at time t_{sync}. That is, p is already well-synchronized at t_{sync} but does not know this fact.

Theorem 5 (First Synchronization via Join). *Let p be an obedient process that gets up at time $t_{up} \geq \sigma_{first}^0$ in a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes running the algorithm of Figure 4, and assume that $k = C_{\max}(t_{up} - \tau^- - \epsilon) + 1 \geq 1$ for any infinitesimally small ϵ. Then, there is some $t_{sync} \leq t_{up} + 2\tau^+$ with $\ell = C_p(t_{sync}) \geq k-2$, such that $p \in \mathcal{P}_{up}(\min\{\ell, k\})$ and*

$$C_{\max}(t) - C_p(t) \leq D_{MCB} \qquad (15)$$

for all times $t \geq t_{sync}$.

Proof. Since $k = C_{\max}(t_{up} - \tau^- - \epsilon) + 1 \geq 1$, it follows that $\sigma_{first}^{k-1} \geq t_{up} - \tau^-$ and hence $p \in \mathcal{P}_{up}(k)$ by Lemma 4. Consequently, k is the first tick that is observed comprehensively by p. We will show that p gets sufficiently many messages to set its clock at least to $\ell \geq k-2$ at some time t_{sync} with $\sigma_p^{\ell-1} \leq t_{sync} \leq t_{up} + 2\tau^+$, and that $p \in \mathcal{P}_{up}(\ell)$ and (15) hold from this time on.

Since p sends its *join* message at time t_{up}, any obedient process receives it within $\boldsymbol{I} = [t_{up} + \tau^-, t_{up} + \tau^+]$. By time $t_{up} + 2\tau^+$, all the responses from correct processes must have arrived at p. We have to distinguish 3 cases:

Case (a): If $\sigma_{first}^k \leq t_{up} + \tau^+$, Lemma 10 reveals $t_{sync} \leq t_{up} + \tau^+ + \varepsilon \leq t_{up} + 2\tau^+$ and hence confirms our lemma for $\ell = k$. So let us assume that $\sigma_{first}^k > t_{up} + \tau^+$, which implies that only σ_{first}^{k-1} could occur within $\boldsymbol{I}$.

Case (b): If $\sigma_{first}^{k-1} \notin \boldsymbol{I}$, the value of $C_{\max}$ does not increase within $\boldsymbol{I}$, i.e., $k' = C_{\max}(t_{up} + \tau^-) = C_{\max}(t_{up} +$

τ^+) with $k' = k$ or $k' = k-1$, depending upon whether (b.1) $\sigma_{first}^{k-1} < t_{up} + \tau^-$ or (b.2) $\sigma_{first}^{k-1} > t_{up} + \tau^+$ (the latter is only possible for $k \geq 2$ due to $t_{up} \geq \sigma_{first}^0$). Lemma 13 assures that there is a set $\mathcal{P}$ of at least $f_\ell^s + f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes at time $t_{up} + \tau^-$ that last sent *(echo, k')* or *(echo, k' − 1)*. Since correct processes send *echo*-messages with increasing index only, and k' is the maximum clock value throughout $\boldsymbol{I}$ and hence the maximum possible index, all the processes in $\mathcal{P}$ last sent *(echo, k')* or *(echo, k' − 1)* at any time $t \in \boldsymbol{I}$. Since at most f_ℓ^s of the processes in $\mathcal{P}$ may have lost p's *join* due to send link failures, there are at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ *(echo, k')* or *(echo, k' − 1)* among the responses on p's *join*-message. Recalling the at most f_ℓ^r receive link failures, it follows that there is some time $\sigma_p^{k'-2} \leq t_{sync} \leq t_{up} + 2\tau^+$ with $|\mathcal{V}_p^{(k'-1)'}(t_{sync})| \geq f_\ell^{ra} + f_a + f_s + 1$. This triggers the catch-up rule (`line 18`) that causes $C_p(t_{sync}) \geq k' - 1$, if this has not happened earlier. Moreover, by time t_{sync}, we can be sure that the responses to *join* recovered all but at most f_ℓ^s tick $k'-1$ messages that might have been missed due to late booting. Hence, at time t_{sync}, the situation at p is equivalent to $p \in \mathcal{P}_{up}(k'-1)$. Note that the up to f_ℓ^s missing *(echo, k' − 1)* or *(echo, k')* originating from lost *join* messages do not matter for the pivotal weak relay (P3W) property, since our n is larger by f_ℓ^s than the n of Theorem 1. Lemma 9 thus guarantees that p is within precision D_{MCB} for all $t \geq t_{sync}$ and hence (15).

Case (c): Finally, if $\sigma_{first}^{k-1} \in \boldsymbol{I}$, the value of $C_{\max}$ increases from $k-1$ to k somewhere within $\boldsymbol{I}$. Consider the at least $f_\ell^s + f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ correct processes that last sent *(echo, k)* or *(echo, k − 1)* according to Lemma 13 at time σ_{first}^{k-1}. If p's *join*-message reached one of those processes q at some time $t \geq \sigma_{first}^{k-1}$, its response message is guaranteed to be *(echo, k − 1)* or *(echo, k)* by the same reasoning as before. If, on the other hand, p's *join*-message reached q at some time $t < \sigma_{first}^{k-1}$, there are two possibilities: If the index of the last sent *echo*-message of q was already $k-1$ at time t, the response message is *(echo, k − 1)* as required. Otherwise, the response message is some *(echo, k − x)* with $x \geq 2$. However, by time σ_{first}^{k-1}, process q must have changed its state such that it last sent *(echo, k)* or *(echo, k − 1)* according to Lemma 2, which means that this message must already be in transit and reaches p by $\sigma_{first}^{k-1} + \tau^+ \leq t_{up} + 2\tau^+$. Like above, it hence follows that there is some time $\sigma_p^{k-2} \leq t_{sync} \leq t_{up} + 2\tau^+$ with $|\mathcal{V}_p^{(k-1)'}(t_{sync})| \geq f_\ell^{ra} + f_a + f_s + 1$, which triggers the catch-up rule (`line 18`) and causes $C_p(t') \geq k-1$, if this has not happened earlier. By the very same reasoning as in case (b) above, we can claim that the *join*-protocol established a situation equivalent to $p \in \mathcal{P}_{up}(k-1)$, and that hence Lemma 9 implies (15) as required. This eventually completes the proof of Lemma 5. □

Equipped with Theorem 5, it is not difficult to bound the time it takes for all obedient processes to become active after the $n - f_a - f_s - f_o - f_c$-th correct process got up. We need some additional properties related to Theorem 1 for this purpose, however, which apply only when at least $n - f_\ell^s - f_a - f_s - f_o - f_c$ processes are up in a system with $n \geq f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes.

Theorem 6 (Clock Synchronization Properties). *In a system with $n \geq f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes, the algorithm from Figure 4 achieves*

- *P1F Uniform Full Correctness. If at least $n - f_\ell^s - f_a - f_s - f_o - f_c$ correct processes set their clocks to $k \geq 0$ by time t, then every obedient process $\in \mathcal{P}_{up}(k)$ sets its clock at least to $k+1$ and becomes active by time $t + 2\tau^+$.*
- *P3 Uniform Relay. Suppose there are at least $n - f_\ell^s - f_a - f_s - f_o - f_c$ correct processes among the processes $\in \mathcal{P}_{up}(k-1)$. If any obedient process sets its clock to $k > 0$ at time t, then every obedient process $\in \mathcal{P}_{up}(k-1)$ also sets its clock at least to k and becomes active by time $t + \tau^+ + \varepsilon$.*

Proof. Uniform Full Correctness. According to Lemma 2, all of the at least $n - f_\ell^s - f_a - f_s - f_o - f_c \geq 2f_\ell^{ra} + 2f_\ell^r + 2f_a + 2f_s + f_o + f_c + 1$ correct processes must have sent either *(init, k)* or *(echo, k)* by time t. Hence, there must be at least $f_\ell^{ra} + f_\ell^r + f_a + f_s + 1$ processes that all sent either *(init, k)* or *(echo, k)*. Every correct process thus achieves sufficient evidence (`line 10`) or catches up to k (`line 18`) by time $t + \tau^+$ and hence sends *(echo, k)*, if it has not already done so. In the latter case, item (1) of Lemma 2 secures that *(echo, k)* has already been emitted. Taking into account the at most f_ℓ^r receive link failures, the *(echo, k)* perception vector of every obedient process $p \in \mathcal{P}_{up}(k)$ at time $t + 2\tau^+$ hence satisfies $|\mathcal{V}_p^{k'}(t + 2\tau^+)| \geq n - f_\ell^s - f_\ell^r - f_a - f_s - f_o - f_c$. Consequently, p sets its clock to $k+1$ by the third **if** (`line 13`) in Figure 4, if it had not already done so. Note carefully that the fifth **if** is executed by time $t + 2\tau^+$ if p has already reached a clock value $\geq k+1$, which could have happened via the catch-up rule. Hence, switching to active mode is guaranteed to happen in this case as well.

Uniform Relay. Assume that $k = C_{\max}(t)$. From Weak Relay (P3W), it follows that every obedient process $\in \mathcal{P}_{up}(k-1)$ sets its clock to at least $k-1$ by time $t+\varepsilon$ since it got *(echo, k − 1)* from at least $f_\ell^{ra} + f_a + f_s + 1$ correct processes. Let q be any of the at least $n - f_\ell^s - f_a - f_s - f_o - f_c$ correct processes in $\mathcal{P}_{up}(k-1)$. If $C_q(t+\varepsilon) \geq k$, it must have sent *(echo, k − 1)* according to Lemma 2. If, on the other hand, $C_q(t+\varepsilon) \leq k-1$, then q must have achieved sufficient evidence or have catched-up by time $t + \varepsilon$ due to (P3W). It follows that every of the $n - f_\ell^s - f_a - f_s - f_o - f_c$ correct processes sent *(echo, k−1)* by time $t+\varepsilon$, which means that every obedient process p receives sufficiently many messages to set its clock to k by time $t + \varepsilon + \tau^+$ via the third **if** (`line 13`). Switching to active mode is guaranteed by the same reasoning as in the proof of (P1F) above.
If $k < C_{\max}(t)$, some obedient process must have set its clock to $k = C_{\max}(t')$ at some time $t' < t$, before the

process assumed in (P3) did so. We showed above that all obedient processes must set their clocks to k by time $t' + \tau^+ + \varepsilon$, so all obedient processes must also set their clocks to k by time $t + \tau^+ + \varepsilon$. □

Now we are ready for our Theorem 7, which bounds the time it takes for all obedient processes to become active after the last correct process got up, and shows that the system is guaranteed to make progress afterward.

Theorem 7 (Initialization Time). *Let $t_{up} \geq \sigma^0_{first}$, with $k = C_{\max}(t_{up} - \tau^- - \epsilon) + 1 \geq 1$ for any infinitesimally small ϵ, be the time when the $n - f_a - f_s - f_o - f_c$-th correct process gets up in a system of $n \geq 2f^s_\ell + 2f^{ra}_\ell + 2f^r_\ell + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes running the algorithm of Figure 4. All obedient processes that got up by time t_{up} switch to active by time $t_{up} + \Delta_{init}$ and progress to at least k by time $t_{up} + \Delta_{init} + 2\tau^+$, where $\Delta_{init} = 3\tau^+ + \varepsilon$.*

Proof. Let p be the $n - f_a - f_s - f_o - f_c$-th correct process that gets up at t_{up}. Theorem 5 ensures that it gets synchronized with $C_{\max}$—and hence with all other processes that are up at t_{up}—within D_{MCB} at time $t_{sync} \leq t_{up} + 2\tau^+$ and satisfies $\ell_p = C_p(t_{sync}) \geq k - 2$ as well as $p \in \mathcal{P}_{up}(\min\{\ell_p, k\})$. Obviously, $q \in \mathcal{P}_{up}(\min\{\ell_q, k\})$ and precision D_{MCB} also holds for obedient processes q that booted earlier, including the early starters. Abbreviating $t' = t_{sync}$ and $k' = C_{\max}(t') \geq k - 1$, we will first show that every obedient process q sets its clock to at least $k' \geq k - 1$ and becomes active by time $t' + \tau^+ + \varepsilon \leq t_{up} + \Delta_{init}$. If already $C_q(t') \geq k'$, nothing remains to be done. In the other case $C_q(t') \leq k' - 1$, we can apply relay (P3) in Theorem 6 since $q \in \mathcal{P}_{up}(k' - 1)$ due to $q \in \mathcal{P}_{up}(\min\{\ell_q, k\})$ and $\ell_q \leq C_q(t') \leq k' - 1$. Hence, q must set its clock to at least k' and switch to active by time $t' + \tau^+ + \varepsilon$ as asserted.
Using full correctness (P1F), it is easy to show that every obedient process q reaches at least clock value $k' + 1 \geq k$ by time $t' + 3\tau^+ + \varepsilon$. Every process q has already reached clock value k', so obviously $q \in \mathcal{P}_{up}(k')$. Hence, q sets its clock to at least $k' + 1$ within another $2\tau^+$. This eventually completes the proof of Theorem 7. □

We still have to consider very late processes, which complete booting after the $n - f_a - f_s - f_o - f_c$-th correct one. The following Theorem 8 gives a bound upon the time until such a process becomes active. Since we have already guaranteed progress here, this is equivalent to integrating processes into running systems.

Theorem 8 (Integration Time). *Suppose that some obedient process gets up at time t_{up} after the $n - f_a - f_s - f_o - f_c$-th correct one. Then, p switches to active by time $t_{up} + \Delta_{init}$ with $\Delta_{init} = 3\tau^+ + \varepsilon$.*

Proof. The proof of Theorem 7 does not depend upon p being the $n - f_a - f_s - f_o - f_c$-th correct process, but could be any obedient process booting after the $n - f_a - f_s - f_o - f_c$-th correct one. Hence, it can be re-used literally here. □

7 Clock Synchronization in Normal Mode

In this section, we will explore the properties of our algorithm during normal mode, where at least $n - f_a - f_s - f_o - f_c$ correct processes are guaranteed to be active. According to Theorem 7, normal mode is entered by time $t_{norm} \leq t_{up} + \Delta_{init}$, where t_{up} is the time when the $n - f_a - f_s - f_o - f_c$-th correct process gets up. We know already from Theorem 5 and Lemma 12 that precision D_{MCB} resp. the upper envelope bound of the accuracy property (A) holds for all obedient active processes, whether normal mode has been reached or not. We will now show that all those processes are also guaranteed to make progress, i.e., obey the lower envelope bound in the accuracy property (A) in normal mode.

The following two preliminary lemmas are building blocks for the lower envelope bound given in Lemma 16.

Lemma 14. *Let p be an obedient process that sets its clock to k at time t in normal mode. Then, p sets its clock to $k + 1$ by time $t + 3\tau^+ + \varepsilon$.*

Proof. The proof uses the properties (P1F) and (P3) from Theorem 6. The relay property (P3) guarantees that all obedient processes set their local clocks to k by time $t' = t + \tau^+ + \varepsilon$. Full correctness (P1F) guarantees that all obedient processes, including p, must set their clocks to $k + 1$ by time $t' + 2\tau^+$. Hence p sets its clock by time $t + 3\tau^+ + \varepsilon$. □

Lemma 15 (Slowest Progress). *Let p be the last obedient process that sets its clock to k at time t in normal mode. Then no obedient process $\in \mathcal{P}_{up}(k)$ can have a smaller clock value than $k' > k$ at time $t + 2\tau^+(k' - k)$.*

Proof. By induction on $l = k' - k$. For $l = 1$, Lemma 15 is identical to (P1F) from Theorem 6 and therefore true. Assume that the last obedient process sets its clock to $k + l$ at time $t + 2\tau^+ l$ for some l. Note that all correct processes have set their clocks to $k + l$ by then. According to (P1F), every obedient process must set its clock to $l + 1$ by time $t + 2\tau^+ l + 2\tau^+ = t + 2\tau^+(l + 1)$. Hence, Lemma 15 is true for $l + 1$ as well. □

The lower bound result in the following Lemma 16 exploits the fact that sufficiently many processes are available in normal mode to ensure that relay (P3) and full correctness (P1F) of Theorem 6 can be applied without restrictions.

Lemma 16 (Lower Envelope Bound). *For a system of $n \geq 2f^s_\ell + 2f^{ra}_\ell + 2f^r_\ell + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes running the algorithm of Figure 4 after normal mode is reached at some time t_{norm}, every active obedient process satisfies $\frac{t_2 - t_1}{2\tau^+} - 3 + \frac{1}{\Theta} < C_p(t_2) - C_p(t_1)$ for all times $t_2 \geq t_1 \geq t_{norm}$.*

Proof. Let $C_p(t_1) = k + 1$ and $C_p(t_2) = k + l + 1$ for some $l \geq 0$. Process p has set its clock to $k + 1$ at time $\sigma^k_p \leq t_1$ and to $k + l + 1$ at time $\sigma^{k+l}_p \leq t_2 < \sigma^{k+l+1}_p$. Hence, $t_2 - t_1 < \sigma^{k+l+1}_p - \sigma^k_p \leq \sigma^{k+l}_p - \sigma^k_p + 3\tau^+ + \varepsilon$

according to Lemma 14[19] and thus $t_2 - t_1 - 3\tau^+ - \varepsilon < \sigma_p^{k+l} - \sigma_p^k$. Using relay (P3) in Theorem 6 in conjunction with Lemma 15, it follows that $\sigma_p^{k+l} - \sigma_p^k \leq 2l\tau^+ + \tau^+ + \varepsilon$ and hence $t_2 - t_1 - 3\tau^+ - \varepsilon < 2l\tau^+ + \tau^+ + \varepsilon$. Since $l = C_p(t_2) - C_p(t_1)$, our lower envelope bound evaluates to $\frac{t_2-t_1}{2\tau^+} - 3 + \frac{1}{\Theta} < C_p(t_2) - C_p(t_1)$ as asserted. □

Besides the lower envelope bound, there is also a possible improvement of precision during normal mode. In degraded mode, precision D_{MCB} essentially follows from weak relay (P3W) in Theorem 1. In normal mode, some specific precision D'_{MCB} also follows from relay (P3) in Theorem 6. It surpasses (P3W) in that it achieves tighter synchronization, but it is worse in that the latency until synchronization happens is greater. The following Theorem 9 provides the resulting D'_{MCB}.

Theorem 9 (Precision in Normal Mode). *In a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes, the algorithm of Figure 4 achieves $C_p(t) - C_q(t) \leq D'_{MCB}$ for all obedient processes p and q in normal mode $t \geq t_{norm}$. $D'_{MCB} = \lfloor \Theta + \frac{1}{2} \rfloor$.*

Proof. Our proof follows the proof of Lemma 8, except that relay (P3) is used instead of weak relay (P3W). Suppose some active obedient process p has clock value $k \geq 1$ at real-time t_p. Since progress is guaranteed in normal mode, $\sigma_{first}^k < \infty$. By relay (P3) in Theorem 6, all obedient processes $\in \mathcal{P}_{up}(k)$, and hence also p, must set their clocks to at least $k+1$ by time $\sigma_{first}^k + \tau^+ + \varepsilon$. This implies

$$\sigma_{first}^k \geq \sigma_p^{k-1} - \tau^+ - \varepsilon. \tag{16}$$

Let $t = \sigma_p^k$ denote the time when p sets its clock to k, and $t' = t - \tau^+ - \varepsilon$. In order to compute $C_{\max}(t)$, we use Lemma 6 and our knowledge of $C_{\max}(t - \tau^+ - \varepsilon) \leq C_{\max}(\sigma_{first}^k) = k+1$ due to monotonicity of $C_{\max}$ in conjunction with (16): From Lemma 6, we infer that $C_{\max}(t) \leq \lfloor \frac{t-t'}{2\tau^-} \rfloor + I_\sigma(t') + C_{\max}(t')$. If $\sigma_{first}^k = t'$ such that t' is synchronized with $C_{\max}$, then $C_{\max}(t') = k+1$ and $I_\sigma(t') = 0$ according to Definition 6. If, on the other hand, $\sigma_{first}^k > t'$, then $C_{\max}(t') = k$ and $I_\sigma(t') = 1$. In both cases, $I_\sigma(t') + C_{\max}(t') = k+1$ and hence $C_{\max}(t) \leq \lfloor \frac{\tau^+ + \varepsilon}{2\tau^-} \rfloor + k + 1 \leq \lfloor \Theta + \frac{1}{2} \rfloor + k$. As $C_{\max}(t_p) \leq C_{\max}(t)$ since $t_p \leq t$ and $C_p(t_p) = k$, the precision D'_{MCB} given in our theorem follows. □

When comparing our two bounds on precision—which are of course both valid during normal mode—we discover a "break even" at $\Theta = 2$. Only for very small uncertainties ($\tau^+ < 2\tau^-$) D'_{MCB} and hence the third **if** (`line 13`) guarantees a better precision. In systems with larger uncertainties, our first bound D_{MCB}—enforced by the fourth **if** (`line 18`)—is better. Our algorithm hence satisfies the accuracy property (A) according to the following Theorem 10.

Theorem 10 (Envelope Condition). *In a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 3f_s + 2f_o + 2f_c + 1$ processes, the algorithm of Figure 4 achieves*

$$\frac{t_2-t_1}{2\tau^+} - 3 + \frac{1}{\Theta} < C_p(t_2) - C_p(t_1) < \frac{t_2-t_1}{2\tau^-} + \min\{D_{MCB}, D'_{MCB}\} + 1$$

for all obedient processes p and all times $t_2 \geq t_1$ during normal mode.

Proof. The lower envelope bound follows literally from Lemma 16. The upper envelope bound is obtained from a slightly modified version of Lemma 12, which exploits the fact that D_{max} can be replaced by $\min\{D_{MCB}, D'_{MCB}\}$ in normal mode. □

8 Conclusions

We described and analyzed a clock synchronization algorithm for partially synchronous systems with unknown bounds $[\tau^-, \tau^+]$ upon delays, which works during system startup and tolerates 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 completely time- and timer-free and hence an asynchronous algorithm. Its timing properties hence solely "emerge" [11] 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 $\Theta = \tau^+/\tau^-$. In terms of assumption coverage, this is an advantageous property [15,16].

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.
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. Ariel Daliot, Danny Dolev, and Hanna Parnas. Linear time byzantine self-stabilizing clock synchronization. In *Proceedings of the 7th International Conference on Principles of Distributed Systems*, Dec 2003. to appear.
4. 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.

[19] Note that $\sigma_p^k \leq t_1$ implies that we might consider the last round switching time before t_{norm} here. Since (P3) has already been applied once before t_{norm} in the proof of Theorem 7, however, using Lemma 14 is legitimate also in this case.

5. Shlomi Dolev and Jennifer L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. In *Proc. of the 2nd Workshop on Self-Stabilizing Systems*, May 1995.
6. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.
7. Christof Fetzer and Flaviu Cristian. An optimal internal clock synchronization algorithm. In *Proceedings 10th Annual IEEE Conference on Computer Assurance*, Gaithersburg, MD, June 1995.
8. 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.
9. 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.
10. Vassos Hadzilacos and Sam Toueg. Fault-tolerant broadcasts and related problems. In Sape Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 2nd edition, 1993.
11. 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.
12. 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.
13. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
14. Gérard Le Lann. Certifiable critical complex computing systems. In K. Duncan and K. Krueger, editors, *Proceedings 13th IFIP World Computer Congress 94*, volume 3, pages 287–294. Elsevier Science B.V. (North-Holland), 1994.
15. 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).
16. Gérard Le Lann and Ulrich Schmid. How to maximize computing systems coverage. Technical Report 183/1-128, Department of Automation, Technische Universität Wien, April 2003.
17. Barbara Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, 6:211–219, 1993.
18. Jennifer Lundelius-Welch and Nancy A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.
19. Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.
20. Paul S. Miner. Verification of fault-tolerant clock synchronization systems. *NASA Technical Paper 3349*, November 1993.
21. 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.
22. Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, October 1990.
23. 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.
24. 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.
25. Ulrich Schmid. Orthogonal accuracy clock synchronization. *Chicago Journal of Theoretical Computer Science*, 2000(3):3–77, 2000.
26. 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.
27. Ulrich Schmid and Christof Fetzer. Randomized asynchronous consensus with imperfect communications. In *22nd Symposium on Reliable Distributed Systems (SRDS'03)*, pages 361–370, October 6–8, 2003.
28. Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, March 1997.
29. Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing*, 14(2):101 – 111, May 2001.
30. Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization with optimal precision. *Information and Computation*, 186(1):36–77, October 2003.
31. 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).
32. 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.
33. Fred B. Schneider. Understanding protocols for byzantine clock synchronization. Technical Report 87-859, Cornell University, Department of Computer Science, August 1987.
34. B. Simons, Jennifer L. Welch, and Nancy Lynch. An overview of clock synchronization. In *Fault-Tolerant Distributed Computing*, LNCS 448, pages 84–96, 1990.
35. T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.
36. T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.
37. 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.
38. Paulo Verı́ssimo, Antonio Casimiro, and Christof Fetzer. The timely computing base: Timely actions in the pres-

ence of uncertain timeliness. In *Proceedings IEEE International Conference on Dependable Systems and Networks (DSN'01 / FTCS'30)*, pages 533–542, 2000.
39. Chris J. Walter and Neeraj Suri. The customizable fault/error model for dependable distributed systems. *Theoretical Computer Science*, 290:1223–1251, October 2002.
40. Josef Widder. Booting clock synchronization in partially synchronous systems. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC'03)*, volume 2848 of *LNCS*, pages 121–135. Springer Verlag, October 2003.