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

Research Report 49/2005, Technische Universität Wien, Institut für Technische Informatik, 2005 November 11, 2008

Achieving Synchrony without Clocks

Josef Widder, Ulrich Schmid

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

Draft: November 11, 2008

Supported by the FWF project Theta *(proj. no. P17757-N04) and the BM:vit FIT-IT project* DCBA *(proj. no. 808198).*

Summary. We present a novel partially synchronous system model, which augments the asynchronous model by a (possibly unknown) bound Θ on the ratio of longest and shortest end-to-end delays of messages simultaneously in transit. An upper bound on those delays need not exist, however, and even Θ may hold only after some unknown global stabilization time. Θ-algorithms are fully message-driven and do not have access to bounded drift local clocks, which makes them particularly suitable for VLSI Systems-on-Chip, for example. In this model, we provide a simulation of (eventually achieved) lock-step rounds, which even works in the presence of Byzantine failures. It follows that most problems in distributed computing have a solution in our model: Using the basic consensus algorithm for partially synchronous systems by Dwork, Lynch and Stockmeyer (1988), for example, Byzantine consensus can be solved. We also introduce a timing transformation technique that facilitates simple correctness proofs and performance analyses of Θ-algorithms, and provide a detailed relation of the Θ-model to other partially synchronous system models.

Key words: computing models – fault-tolerant distributed algorithms – partially synchronous systems – clocks and time

1 Introduction

Exploring the inherent power and limitations of synchrony assumptions in distributed computing models is a long-standing goal of distributed computing research. Synchrony assumptions relate the occurrence times of events in a distributed system to each other, and several seminal papers e.g. on partially synchronous systems [13, 15] and unreliable failure detectors [11] have contributed much to the understanding of the relationship between synchrony conditions and solvability of certain distributed computing problems. The most heavily researched problem here is consensus, which is the ability of a set of distributed processes to agree on a common decision value, despite failures: Consensus is impossible to solve deterministically in asynchronous systems, i.e., in systems with unbounded delays and unbounded relative process speeds [21], but can be solved when sufficiently strong timing assumptions are added [13].

Obviously, the weaker the properties to be added to the asynchronous model, the higher is the probability that these properties can indeed be guaranteed during the operation of a real system. Finding the weakest system model for solving consensus has hence always been a very active branch of distributed computing research; see e.g. [2, 4, 8, 9, 11–13, 15, 20, 25–27, 31, 34, 41, 47, 51].

In the existing literature, synchrony conditions have been considered primarily in the form of explicit timing assumptions, i.e., assumptions that relate occurrence times of events to real-time. For example, the classic (non lock-step) synchronous model rests on known upper and lower bounds on message transmission delays and computing step times, in conjunction with bounded drift clocks. Using a clock synchronization algorithm like the one of [32], lock-step rounds can be simulated easily in the synchronous model, which in turn makes synchronous consensus algorithms applicable. Similarly, the classic partially synchronous models of [15] assume a bound on message transmission delays Δ, but replace the computing step time bounds by a bound Φ on the relative speed of the processes. Eventual lock-step rounds can be simulated in these models, which in turn allows to solve consensus by means of the algorithms proposed in [15]. Other partially synchronous models [2,4,8,9,12,20,41,47] require bounded drift local clocks, in some way or other, for timing out certain events and triggering computing steps. All the above models hence apply only to systems that satisfy certain timing properties, which are expressed via explicit real-time bounds (unit "seconds").

For restricted failure assumptions, in particular, process crashes, a time-free alternative is provided by algorithms that are based on unreliable failure detectors [11]: Fairly weak failure detectors, like the eventually strong failure detector $\diamond\mathcal{S}$ or the leader oracle Ω, are known to

Correspondence to: Josef Widder

be sufficient for solving consensus, and the perfect failure detector $\mathcal{P}$ even allows to establish a round structure in a time-free way [23]: Every process can terminate a round when it received a message by every process that does not appear on its suspect list. Hence, for certain problems, synchrony assumptions can be replaced by assumptions related to the ability of the processes to detect crashes.

Interestingly, failure detection itself can also be done without resorting to explicit timing assumptions, i.e., without bounds on message delays, computing step times and bounded drift clocks: In [35,36,38], it has been shown that all important failure detectors can be implemented in systems that provide a priori guarantees of certain message order properties. It is not known yet, however, whether such order properties can be enforced without resorting to timing assumptions. Similarly, in [25,31,51], we have shown that failure detectors can be implemented in asynchronous systems augmented with the assumption that the *speed ratio* of concurrent distributed computations is bounded: Our simple Θ-Model just assumes that the ratio of the (unknown) maximum and minimum values of the *end-to-end delay* of all messages simultaneously in transit, denoted τ^+ and τ^-, respect the bound $\tau^+/\tau^- \leq \Theta$.

In contrast to classic synchrony assumptions, the Θ-assumption does not involve absolute time bounds: Since τ^+ and τ^- (unlike their ratio) are unknown (we will show in this paper that they can even vary arbitrarily with time, and hence do not satisfy any given bound), Θ-algorithms are *time-free* in the sense that neither code nor variables contain absolute timing values (unit "seconds"). In addition, they are purely *message-driven* [10] in the sense that computing steps are not triggered by the passage of time but only by the reception of messages. Consequently, Θ-algorithms are easily portable and do not need bounded drift clocks, which simplifies design and implementation and improves the model coverage. The price to be paid, however, is the need to continously send (some) messages; this creates some system load and decreases the coverage (as the system turns mute in case of total message loss). Whether the advantages of relaxed synchrony requirements outweigth the disadvantages of the message-driven execution is of course very application-specific, but there are at least some applications like VLSI Systems-on-Chip where this is indeed the case, see Section 5 for more information.

Accomplishments: In this paper, we weaken the synchrony assumptions of the simple Θ-Model further, and devise a Byzantine fault-tolerant lock-step round simulation for the resulting models. More specifically:

1. In Section 2, we introduce four different versions of the Θ-Model, denoted $\Theta, ?\Theta, \diamond\Theta$, and $\diamond?\Theta$, depending on whether Θ is known/unknown and holds perpetually/eventually. Moreover, every model comes in a static and a dynamic variant, according to whether the timing behavior is time-invariant or time-variant. The simple Θ-Model from [31,51] corresponds to the static Θ-Model with known Θ holding perpetually.
2. In Section 3, we give a time transformation technique, which proves formally that any dynamic Θ-Model is equivalent to its static counterpart in terms of solvability of problems and proofs. Algorithms designed for the dynamic Θ-Models can hence be proved correct in the simpler static models.
3. In Section 4, for each of the models, we present an algorithm that, from some time on, simulates lock-step rounds even in the presence of Byzantine failures. In conjunction with the consensus algorithm from [15], for example, this yields solutions for Byzantine consensus in any Θ-Model.
4. In Section 5, we discuss some general aspects of the applicability of the Θ-Model and its limitations. We will also show how to reduce the system load created by Θ-algorithms.
5. In Section 6, we provide a detailed relation of (the synchrony-related properties of) the Θ-Models to the classic partially synchronous models from [11,15], as well as a brief comparison to the other partially synchronous models proposed in literature.

2 The Θ-Model

We consider a system consisting of a finite set of n distributed processes denoted as $p, q, \ldots$, which communicate through a fully connected point-to-point message passing network. Every receiver process can hence be sure of the identity of the sender of a message, although there is no authentication service. Among the n processes there may be $f < n/3$ Byzantine faulty ones; i.e., no assumption is made on the behavior of faulty processes.[1] The communication channels between correct processes must be reliable but not necessarily FIFO.

Every correct process executes an instance of a distributed algorithm and is modeled as a state machine. Its local execution consists of a sequence of atomic, zero-time computing steps, each involving the reception of at least one message, a state transition, and the sending of zero or more messages to a subset of the processes in the system.

Processes do not have access to any local clock and execute message-driven algorithms [10] only: Computing steps at process p are exclusively triggered by the arrival of messages at p here, not by the passage of time as in time-driven execution models like [15]. The very first computing step of a process is triggered by an external "wake-up message"; we assume that this very first step occurs before any message from another process is received.

The Θ-Model does not impose any relation between computing step times and message delays. Instead, send

[1] In this paper we consider Byzantine faults as they encapsulate all other failure modes (as omission, value, or timing failures). However, the definitions in this section can naturally be adapted to more restricted failure modes. In [52], for instance, we show how to extend such definitions to a hybrid failure model, involving several different classes of process and link failures.

and receive computing step times and message transmission delays will be combined into end-to-end delays:

Definition 1 (End-to-end delays). *Let t_s^m denote the real-time when a sender process s sends message m to process r in some computing step, and t_r^m be the real-time when the receive computing step triggered by message m occurs at r. Message m is called* in transit *during the interval $[t_s^m, t_r^m)$, and $\delta_{sr}^m = t_r^m - t_s^m$ denotes the end-to-end delay of message m. If s and r are correct processes, then $0 < \delta_{sr}^m < \infty$ for every message m.*

In asynchronous systems [21], both computing step times and message transmission delays — and hence end-to-end delays — are arbitrary but finite. Consensus cannot be solved deterministically in this model [21]. The simple Θ-Model introduced in [31,51] overcomes this impossibility by restricting the delays between correct processes to respect some unknown upper and lower bound τ^+ and τ^-, respectively, with ratio $\Theta = \tau^+/\tau^-$. This model has proved to be a simple and efficient means for designing and analyzing Θ-algorithms [31, 49–51].

In Definition 2, we extend the simple Θ-Model to a variety of 4 different models, depending on whether Θ is known or unknown[2] and whether the Θ-bound holds *eventually* (after some unknown global stabilization time GST > 0) or *perpetually* (from the very beginning, i.e., GST $= 0$). To distinguish these models from the time-variant dynamic versions introduced later in Definition 3, they will be called *static* in the sequel.

Definition 2 (Static Θ-Models). *Let $\tau^+ < \infty$ and $\tau^- > 0$, with $\tau^- \le \tau^+$ and timing uncertainty $\varepsilon = \tau^+ - \tau^-$, denote the unknown upper and lower bound, respectively, on the end-to-end delays δ_{sr}^m of all messages m which are sent after* GST *from correct process s to correct receiver[3] process r. The static Θ-Models stipulate a bounded* delay ratio *$\Theta = \tau^+/\tau^-$, for some finite real $\Theta \in \mathbb{R}, 1 \le \Theta < \infty$, as follows:*

- *Θ-Model: Θ is known and holds perpetually*
- *?Θ-Model: Θ is unknown and holds perpetually*
- *$\diamond\Theta$-Model: Θ is known and holds eventually*
- *$\diamond$?Θ-Model: Θ is unknown and holds eventually*

The above variants of the Θ-Model hence differ in the knowledge a Θ-algorithm can rely on: In the Θ-Model and in the $\diamond\Theta$-Model, Θ is known a priori and can hence be compiled into the algorithm; Θ holds in every execution right from the start (GST $= 0$, in the Θ-Model) or eventually (after some unknown time GST > 0, in the ?Θ-Model). By contrast, in the ?Θ-Model and in the $\diamond$?Θ-Model, one only knows that in every execution there is such a value Θ that holds right from the start (GST $= 0$, in the ?Θ-Model) or eventually (after some unknown time GST > 0, in the $\diamond$?Θ-Model); since the value Θ is unknown, however, it cannot be compiled into the algorithm.

Remarks.

1. Each of the models above corresponds naturally to one of the classic partially synchronous models [11, 15], cf. Definition 8.
2. Our reliable network assumption for correct processes requires that even messages that were sent before GST are eventually received. This assumption is vital for any message-driven algorithm, which might turn mute in the presence of message loss between correct processes.[4]
3. Although neither τ^-, τ^+ nor the timing uncertainty $\varepsilon = \tau^+ - \tau^-$ can appear in a Θ-algorithm, those quantities can of course be used as (unvalued) variables in the *analysis* of such algorithms.

The time-varying *dynamic Θ-Models* introduced in Definition 3 below restrict the admissible end-to-end delays in a considerably weaker way. In particular, they do not rest on (unknown) lower and upper bounds for all messages that are sent in an execution, but rather on a bounded ratio $\Theta(t) = \tau^+(t)/\tau^-(t) \le \Theta$ of the longest $\tau^+(t)$ and shortest $\tau^-(t)$ end-to-end delay of those messages that are simultaneously in transit at time t, recall Definition 1. In case of the dynamic variants of the $\diamond\Theta$-Model and the $\diamond$?Θ-Model, we will assume that those restrictions hold after GST only.

Obviously, the resulting dynamic Θ-Models are considerably weaker than the static Θ-Models, as for some fixed Θ, a larger set of admissible executions exists. On the down side, dynamic models render the analysis of Θ-algorithms much more complicated. In Section 3, however, we will introduce a time transformation technique that allows to prove correct and analyze Θ-algorithms designed for dynamic models in the much simpler static models.

Definition 3 (Dynamic Θ-Model). *Let $\mathcal{M}(t)$ be the set of all messages m that are in transit between correct processes at real-time t, with the additional restriction $t_s^m >$ GST for all $m \in \mathcal{M}(t)$. Let for all times t and for some arbitrarily small $\eta > 0$, $\tau^-(t) \ge \eta > 0$ be a non-negative function that builds a lower envelope on end-to-end delays of all messages in $\mathcal{M}(t)$ at real-time t, such that for any time t with $|\mathcal{M}(t)| > 0$ it holds that $\tau^-(t) \le \min\{\delta_{sr}^m\}$ for all $m \in \mathcal{M}(t)$ and correct processes r and s. Similarly, we define an upper envelope function $\tau^+(t) \ge \tau^-(t)$ such that $\max\{\delta_{sr}^m\} \le \tau^+(t) < \infty$ is satisfied for all $m \in \mathcal{M}(t)$. The* delay ratio *$\Theta(t)$ at time t is defined as $\Theta(t) = \tau^+(t)/\tau^-(t)$ and must satisfy $\Theta(t) \le \Theta$ for some finite real $\Theta \in \mathbb{R}, 1 \le \Theta < \infty$, as follows:*

[2] In concordance with [15], the notion of *known* refers to the fact that a parameter has the same finite bounded value in every execution, whereas *unknown* means that in every execution the parameter is finite and bounded but its value may be different in different executions.

[3] To simplify our analysis, we will assume that a process also sends a message to itself in our algorithms. Hence, we also require that $\tau^- \le \delta_{ss}^m \le \tau^+$ for every correct process s. This assumption can of course be dropped if a more refined analysis is conducted.

[4] Clearly, messages to/from faulty processes may always be lost. Moreover, the message-driven algorithm for our hybrid failure model given in [52] also tolerates considerable message loss between correct processes.

- *Θ-Model: Θ is known and holds perpetually*
- *?Θ-Model: Θ is unknown and holds perpetually*
- *$\diamond\Theta$-Model: Θ is known and holds eventually*
- *$\diamond$?Θ-Model: Θ is unknown and holds eventually*

For any real-time t where $\mathcal{M}(t) = \emptyset$, which could happen only (1) during system initialization and before GST *and/or (2) after termination of a terminating algorithm, we define $\tau^-(t) = \tau^+(t) = 1$, such that $\Theta(t) = 1 \leq \Theta$.*

Remarks.

1. We do not restrict the delay of messages exchanged by correct processes when they have been sent before GST, except that they may not be lost, i.e., they have to be received within finite time, recall Remark 2 on Definition 2.
2. We define $\tau^-(t) = 1$ in case of no messages in transit, which simplifies the definition of the timing transformation function $\beta(t)$ in Section 3. This convention has no influence on delay bounds, however, and does in fact not affect normal operation of an algorithm.

3 Equivalence of Dynamic and Static Model

For executions where the delay envelopes in Definition 3 do not vary with real-time t (i.e., are constant functions), the dynamic and the static model are the same. In this section, we will show even more: Suppose that we are given a dynamic model with some bound Θ on the delay ratio. Let the *corresponding* static model be the model according to Definition 2 where the unknown (but fixed) τ^- and τ^+ satisfy the dynamic model's Θ. It will turn out that one can analyze any Θ-algorithm designed for the dynamic model in the corresponding static model and translate the results back.

We will introduce a timing transformation technique for this purpose, which allows to transform time-varying delays into constant delays. It is based on a suitable mapping function β from real-time t to some "stretched" real-time $t' = \beta(t)$, which preserves the order of steps in an execution. Since Θ-algorithms are time-free, they cannot distinguish whether they execute in an "universe" where real-time t or rather t' applies. Consequently, any Θ-algorithm can be analyzed and proved correct in the much simpler static model, and the results can be transformed back to the dynamic model by employing the inverse mapping $\beta^{[-1]}$.

To start our formal treatment, consider the timed execution $\mathcal{E}$ of some Θ-algorithm $\mathcal{A}$ in our dynamic model with delay ratio Θ. We assume that $\mathcal{E}$ runs in a universe $\mathcal{U}_t$ with Newtonian (i.e., continuous) real-time t. As usual, every step $\phi^i(t)$ in $\mathcal{E}$ is assigned its occurrence real-time t (that is not available to the algorithm) for analysis purposes. Now consider the execution $\mathcal{E}'$ of the same algorithm $\mathcal{A}$ in a universe $\mathcal{U}'_t$ with a different notion of real-time t', defined by

$$t' = \beta(t) = \int_0^t \frac{\tau^-}{\tau^-(\nu)}\, d\nu, \tag{1}$$

where τ^- is the (unknown) lower bound of a static model with the same ratio Θ. There is no need to constrain $\tau^-(t)$, except that it must be integrable. Note that if for all times t, $\tau^-(t)$ has its maximal value allowed by Definition 3, $\tau^-(t)$ is a step function — changing its value only when steps are taken — and it is therefore integrable.

Let $\mathcal{E}'$ consist of exactly the same steps as $\mathcal{E}$, except that any step $\phi^i(t)$ occurring at time t in $\mathcal{E}$ occurs as $\phi^i(t')$ at time t' in $\mathcal{E}'$. The following Lemma 1 shows that a process executing $\mathcal{A}$ cannot distinguish $\mathcal{E}$ from $\mathcal{E}'$.

Lemma 1 (Indistinguishability). *Every process has the same local view in the executions $\mathcal{E}$ and $\mathcal{E}'$.*

Proof. It suffices to show that for any two steps $\phi^i(t_i)$ and $\phi^j(t_j)$ occurring at real-times t_i and t_j in $\mathcal{E}$ it holds that the corresponding steps $\phi^i(t'_i)$ and $\phi^j(t'_j)$ in $\mathcal{E}'$ occur at real-times $t'_i < t'_j$ iff $t_i < t_j$. Since Equation (1) integrates a non-negative function $0 < \tau^-/\tau^-(t) < \infty$, according to Definition 3, $\beta(t)$ is continuous and strictly monotonically increasing, such that $t_i < t_j \Rightarrow t'_i = \beta(t_i) < t'_j = \beta(t_j)$. Since the inverse $\beta^{[-1]}(t')$ of a continuous strictly monotonic function is also strictly monotonic, we also have $t'_i < t'_j \Rightarrow t_i = \beta^{[-1]}(t'_i) < t_j = \beta^{[-1]}(t'_j)$. □

Lemma 2 shows that the execution $\mathcal{E}'$ is admissible with respect to the corresponding static model.

Lemma 2 (Admissible Execution). *The execution $\mathcal{E}'$ satisfies Definition 2 of the corresponding static model.*

Proof. We have to show that any message m sent in $\mathcal{E}'$ after GST from correct process s to correct process r has delay $\delta_{sr}^{m\prime}$ with $\tau^- \leq \delta_{sr}^{m\prime} \leq \tau^+ \leq \Theta\tau^-$. Let t_s^m and t_r^m be the send and receive real-times, respectively, of the same message m in $\mathcal{E}$. The corresponding occurrence times in $\mathcal{E}'$ are $t_s^{m\prime} = \beta(t_s^m)$ and $t_r^{m\prime} = \beta(t_r^m)$. Hence, we only have to evaluate $\beta(t_r^m) - \beta(t_s^m)$.

For any time $t \in [t_s^m, t_r^m)$, we have $t_r^m - t_s^m \geq \tau^-(t)$ and $t_r^m - t_s^m \leq \tau^+(t) \leq \Theta\tau^-(t)$ according to Definition 3; recall that m is in transit throughout this interval. This implies $t_r^m - t_s^m \geq \tau^-(t) \geq \frac{t_r^m - t_s^m}{\Theta}$, such that

$$\delta_{sr}^{m\prime} = \beta(t_r^m) - \beta(t_s^m) = \int_{t_s^m}^{t_r^m} \frac{\tau^-}{\tau^-(t)}\, dt$$

satisfies

$$\delta_{sr}^{m\prime} \leq (t_r^m - t_s^m) \cdot \frac{\Theta\tau^-}{t_r^m - t_s^m} = \tau^+$$

$$\delta_{sr}^{m\prime} \geq (t_r^m - t_s^m) \cdot \frac{\tau^-}{t_r^m - t_s^m} = \tau^-$$

by monotonicity of integrals as required. □

From the above, it follows that the dynamic model and the static model essentially have the same expressive power: The dynamic model, where delays need not be bounded, can be mapped to a partially synchronous model with fixed but unknown upper and lower delay bounds. Safety and liveness properties of Θ-algorithms can hence be analyzed in the simple static model, and

translated back to the more "elastic" dynamic model: For properties that do not involve the unknown values τ^-, τ^+ and ε (e.g. our major Theorem 2) this is ensured by Lemma 1. For timing-related properties, we will show below how the results obtained in the static model can be translated back to the dynamic model via the inverse function $\beta^{[-1]}(t')$.

The worst case timing analysis of typical Θ-algorithms [31, 51] in the static model relies on identifying the worst case execution $\mathcal{E}'$ and typically yields performance metrics that are multiples of τ^+ and τ^-. In order to transfer these results, all that is needed is a function that determines the time t in a dynamic execution that corresponds to time t' in the static execution as provided by the transformation Equation (1): When $\tau^-(t)$ is known, it is usually (e.g., in the case of the step function mentioned above) possible to explicitly compute $\beta(t)$ by solving the integral, and to solve $t' = \beta(t)$ for $t = \beta^{[-1]}(t')$. Clearly, $\beta^{[-1]}(t')$ can then be used to map $\mathcal{E}'$ to $\mathcal{E}$: Any step $\phi(t')$ is just mapped to $\phi(t)$ with $t = \beta^{[-1]}(t')$.

Any timing-related property involving some difference $t'_2 - t'_1$ of the occurrence times of two steps $\phi(t'_2)$ and $\phi(t'_1)$ in $\mathcal{E}'$ is hence mapped to $\beta^{[-1]}(t'_2) - \beta^{[-1]}(t'_1)$ in $\mathcal{E}$, which in turn allows to transfer all timing performance metrics from the static model to the dynamic model.

```
0:  VAR k : integer := 0;    // clock value
1:  VAR r : integer := 0;    // round number
2:
3:  /* Initialization */
4:  send (tick 0), round_msg_0 to all;
5:
6:  if received (tick ℓ) from at least f + 1 distinct
    processes, with ℓ > k then
7:      for i := k + 1 to ℓ do
8:          k := i;
9:          action();
10: fi
11:
12: if received (tick k) from at least n − f distinct
    processes then
13:     k := k + 1;
14:     action();
15: fi
16:
17: procedure action();
18: begin
19:     if k = (r + 1)Ξ then
20:         execute lock-step round r's step;
21:         send (tick k), round_msg_{r+1} to all processes;
22:         r := r + 1;
23:     else
24:         send (tick k) to all;
25:     fi
26: end;
```

Fig. 1. Lock-step round simulation for known Θ

4 Lock-Step Round Simulations

4.1 Θ-Model and $\diamond\Theta$-Model

In this section, we introduce a simulation that achieves lock-step rounds both in the Θ-Model and in the $\diamond\Theta$-Model with known Θ, in the presence of up to $f < n/3$ Byzantine faulty processes; analogous simulations for unknown Θ are described in Section 4.2. Lock-step round algorithms proceed in a sequence of rounds $r = 0, 1, \ldots$, each of which consists of sending, receiving and processing round r messages. Processing is done in a single round r computing step at the end of round r, where the round $r + 1$ messages to all processes are sent. Round 0 messages are sent upon initialization. It was shown in [15] that Byzantine consensus can be solved if lock-step rounds can be achieved eventually.

Our simulation is given in Figure 1. Its core is a variant of Srikanth and Toueg's non-authenticated clock synchronization algorithm [45], which ensures that logical clock values — i.e., integer tick values — remain close to each other. The algorithm is started by initializing the local clock to $k := 0$ and sending $(tick\ 0)$ in `line 4`. If a correct process p receives $f + 1$ $(tick\ \ell)$ messages (`line 6`), it can be sure that at least one of those was sent by a correct process; p can hence catch-up to ℓ and send $(tick\ k + 1), \ldots (tick\ \ell)$.[5] If a correct process p receives $n - f \geq 2f + 1$ $(tick\ k)$ messages (`line 12`), it can be sure that at least $f+1$ of those will be received by every correct process — which then executes `line 6` — such that all correct processes will eventually receive $n - f$ $(tick\ k)$ messages within bounded time as well. Thus, when receiving $n - f$ $(tick\ k)$ messages, a process can advance its clock to $k + 1$. We will prove in the sequel that the synchronization properties achieved by this algorithm ensure lock-step rounds after GST.

The execution of the lock-step round r computing step is done in `line 20`, whereas sending of the computed round $r + 1$ message is done in `line 21`, by appending this message to the appropriate clock synchronization message. The idea behind the simulation is straightforward: Lock-step round $0, 1, 2, \ldots$ terminates when the clock value reaches Ξ, $2{\cdot}\Xi$, $3{\cdot}\Xi$, ..., i.e., every Ξ ticks — we will give a bound on Ξ in Theorem 2. The elapsed time between these clock ticks will be such that all correct processes have received the round r messages from all correct processes before they execute their round r computing step.

We start with some definitions of clock values and lock-step round numbers.

Definition 4 (Local Clock Value). *$C_p(t)$ denotes the local clock value of a correct process p at real-time t; σ_p^k, where $k \geq -1$, is the real-time when process p sets its*

[5] Since a process sends its $(tick\ \ell)$ message only after it has sent its $(tick\ \ell - 1)$ message, a practical implementation of the algorithm would not send the messages for the "skipped" ticks, but rather interpret a $(tick\ \ell)$ message as $(tick\ k)$ for all $k \leq \ell$. Making the sending of skipped ticks explicit simplifies the presentation of the algorithm and its analysis, however.

local clock to $k+1$. We use the convention that $C_p(\sigma_p^k) = k+1$.

Note that $\sigma_p^k = \sigma_p^{k+1} = \cdots = \sigma_p^{\ell-1}$ if the clock update happens in `line 6`, since the for-loop is executed within a single computing step.

Definition 5 (Local Round Number). *$R_p(t)$ denotes the local lock-step round number of a correct process p at real-time t; ϱ_p^r, where $r \geq -1$, is the real-time when process p sets its round number to $r+1$ (meaning that it is now in round $r+1$). We use the convention that $R_p(\varrho_p^r) = r+1$.*

Definition 6 (Maximum Local Clock). *$C_{max}(t)$ denotes the maximum of all local clock values of correct processes at real-time t. Further, let $\sigma_{first}^k = \sigma_p^k \leq t$ be the real-time when the first correct process p sets its local clock to $k+1 = C_{max}(t)$.*

Definition 7 (Maximum Local Round). *By $R_{max}(t)$ we denote the maximum of all local lock-step round numbers r of correct processes at real-time t. Further, let $\varrho_{first}^r = \varrho_p^r \leq t$ be the real-time when the first correct process p sets its lock-step round to $r+1 = R_{max}(t)$.*

We start our analysis with the key observation given in Lemma 3. Note that we will conduct our analysis in the static Θ-Models of Definition 2. Lemma 1 and Lemma 2 could be used to translate our results to the dynamic variants, as explained in Section 3.

Lemma 3 (First progress). *The first correct process that sets its clock to $k = C_{\max}(t) > 0$ by time t must do so by `line 12`.*

Proof. By contradiction. Assume that the first correct process p sets its clock to $k = C_{\max}(t)$ at time t by `line 6`. At least one correct process must have sent a message for a tick $\ell \geq k$ to enable the rule at p. Since correct processes only send messages for ticks less or equal their clock value, at least one must already have had a clock value $\ell \geq k$ at instant t. This contradicts p to be the first one that reaches clock value k. □

The following lemma follows immediately from the code of Algorithm 1.

Lemma 4. *For every correct process p and every $r \geq -1$, it holds that $\varrho_p^r = \sigma_p^k$ for $k = (r+1)\Xi - 1$.*

The following theorem introduces the four basic properties of our algorithm. For now, we restrict our attention to the perpetual Θ-Model (GST = 0). Lemma 5 will show that these properties eventually hold also in the $\diamond\Theta$-Model (GST > 0). Recall that $\varepsilon = \tau^+ - \tau^-$ is the timing uncertainty of the end-to-end delay, which obviously satisfies $\varepsilon \leq \tau^+$. Since $\tau^- > 0$ in real systems, ε is actually smaller than τ^+.

Theorem 1 (Clock Synchronization). *In systems of $n \geq 3f+1$ processes adhering to the perpetual Θ-Model, the algorithm given in Figure 1 achieves:*

(P) Progress. *If all correct processes set their clocks at least to k by time t, then every correct process sets its clock at least to $k+1$ by time $t+\tau^+$.*

(U) Unforgeability. *If no correct process sets its clock to k by time t, then no correct process sets its clock to $k+1$ by time $t+\tau^-$ or earlier.*

(Q) Quasi Simultaneity. *If some correct process sets its clock at least to $k > 0$ at time t, then every correct process sets its clock at least to $k-1$ by time $t+\varepsilon$.*

(S) Simultaneity. *If some correct process sets its clock at least to k at time t, then every correct process sets its clock at least to k by time $t+\tau^+ + \varepsilon$.*

Proof. We show the properties separately.

Progress. By assumption, at least $n-f$ correct processes set their clocks at least to k, and hence send (*tick* k) by time t. These messages must be received by all correct processes by time $t+\tau^+$. They thus have set their clocks to k by `line 6` and thus set their clocks to $k+1$ by `line 12`, if they have not already done so.

Unforgeability. Assume by contradiction that a correct process p sets its clock to $k+1$ by time $t+\tau^-$. Correct processes may set their clock by (1) `line 12` or (2) `line 6`.

Assume (1). Process p does so based on $n-f$ (*tick* k) messages by distinct processes, i.e., at least $n-2f$ were sent by correct processes. At least one of these messages was sent by time t, which provides the required contradiction.

Assume (2). Process p does so based on $f+1$ (*tick* $k+1$) messages, i.e., at least one correct process has already clock value $k+1$ by time $t+\tau^-$. The first correct process which has set its clock to $k+1$ must have done so by `line 12` according to Lemma 3. By applying (1), we again derive a contradiction.

Quasi Simultaneity. Let process p be the first correct process to set its clock to $k > 0$. By Lemma 3, p sets its clock to k using `line 12`. This happens based on $n-f \geq 2f+1$ (*tick* $k-1$) messages from distinct processes. At least $f+1$ of these must be sent by correct processes whose messages are received by all correct processes by $t+\varepsilon$. They then set their clocks to $k-1$ by `line 6`, if they have not already done so.

If process p is not the first process which sets its clock to k, then at least one correct processes has done so at time $t' < t$. By the same reasoning as above, all correct processes must set their clocks to $k-1$ by time $t' + \varepsilon < t + \varepsilon$.

Simultaneity. For $k = 0$, (S) follows immediately from initial synchronization (simultaneous booting). For $k > 0$, according to the proof of (Q), all correct processes set their clock at least to $k-1$ and send (*tick* $k-1$) by $t+\varepsilon$. All these messages are received by all correct processes at most τ^+ later, which causes them to set their clocks at least to k by time $t+\tau^+ + \varepsilon$. □

The following Lemma 5 generalizes Theorem 1 to the eventual $\diamond\Theta$-Model. Recall from Section 2 that, before GST, message delays may be arbitrary but there is no message loss.

Lemma 5 (Clock Synchronization for GST > 0). *Let $k_{\mathrm{GST}} = C_{max}(\mathrm{GST}) + 1$. Then (P), (U), (Q), and (S) hold for all clock values k greater than k_{GST}.*

Proof. Since our algorithm is totally time free, the progress part (updating the clocks) of (P) is always satisfied — just the time bound (i.e., $t + \tau^+$) may be violated before GST. It hence follows that there must exist a time $\sigma_{first}^{k_{\mathrm{GST}}-1} > \mathrm{GST}$, and all (*tick* ℓ) messages for $\ell \geq k_{\mathrm{GST}}$ from correct processes are sent after $\sigma_{first}^{k_{\mathrm{GST}}-1}$ and hence after GST. Since those messages — and all subsequent ones — respect our timing assumptions, it follows that all correct processes set their clocks to some value greater than k_{GST} based on messages sent by correct processes after GST. Hence, for $k \geq k_{\mathrm{GST}} + 1$, the proof of Theorem 1 applies literally. □

In order to unify the perpetual case GST = 0 and the eventual case GST > 0, we employ the convention that $k_{\mathrm{GST}} = -1$ if GST = 0.

Lemma 6 (Slowest and Fastest Progress). *For every $R \geq P$, $P > k_{\mathrm{GST}}$ and for any two correct processes p and q,*

$$|\sigma_p^R - \sigma_q^R| \leq \tau^+ + \varepsilon \tag{2}$$

$$\sigma_p^R - \sigma_q^P \geq (R-P)\tau^- - \tau^+ - \varepsilon \tag{3}$$

$$\sigma_p^R - \sigma_q^P \leq (R-P)\tau^+ + \tau^+ + \varepsilon. \tag{4}$$

Proof. Equation (2) follows immediately from the simultaneity property (S), which establishes (3) and (4) for $P = R$ as well. We can hence assume that (3) holds for $k - 1 = R - P - 1 \geq 0$ and have to show that it holds for $k = R - P$ as well. Since process p sets its clock to $R + 1$ at time σ_p^R, at least one correct process p_1 must have sent (*tick* R) by time $\sigma_p^R - \tau^-$ by the unforgeability property (U). Hence $\sigma_p^R - \sigma_{p_1}^{R-1} \geq \tau^-$. By the induction hypothesis, we have $\sigma_{p_1}^{R-1} - \sigma_q^P \geq (R-P-1)\tau^- - \tau^+ - \varepsilon$, from which (3) follows immediately.

For the upper bound, we can assume that (4) holds for $k - 1 = R - P - 1 \geq 0$ and have to show that it holds for $k = R - P$ as well. Since we know that even the last correct process sets its clock to R at some time $\sigma_{\max}^{R-1}$ satisfying $\sigma_{\max}^{R-1} - \sigma_q^P \leq (R-P-1)\tau^+ + \tau^+ + \varepsilon$ by the induction hypothesis, it follows from the progress property (P) that process p must set its clock to $R + 1$ by time $\sigma_{\max}^{R-1} + \tau^+$, from where (4) follows again. □

Theorem 2. *From $k_{\mathrm{GST}} + 1$ on, the algorithm given in Figure 1 with integer $\Xi \geq 3\Theta$ is a correct lock-step round simulation.*

Proof. We have to show that correct lock-step rounds are eventually achieved, i.e., that all round r messages from correct processes are received by every correct process before its round r computing step is taken.

By Lemma 5, (P), (U), (Q), and (S) hold for all $k > k_{\mathrm{GST}}$. Hence, it only remains to show that our algorithm simulates lock-step rounds if these properties hold. Let $k > k_{\mathrm{GST}} + \Xi$ be any integer such that $k = (r+1)\Xi - 1$ for some integer $r \geq 0$. From Lemma 4, we know that every correct process p sends its lock-step round $r+1$ message along with (*tick* $k+1$), i.e., $\varrho_p^r = \sigma_p^k$. The lock-step round simulation is correct if $\varrho_{first}^r - \varrho_q^{r-1} - \tau^+ > 0$ for every correct process q. This follows immediately from using Lemma 6 in

$$\begin{aligned}\varrho_{first}^r - \varrho_q^{r-1} - \tau^+ &= \sigma_{first}^k - \sigma_q^{k-\Xi} - \tau^+ \\ &\geq \Xi\tau^- - 2\tau^+ - \varepsilon \\ &\geq 3\tau^+ - 2\tau^+ - \tau^+ + \tau^- \\ &\geq \tau^- > 0,\end{aligned}$$

completing our proof. □

```
0:  VAR k : integer := 0;    // clock value
1:  VAR r : integer := 0;    // round number
2:
3:  /* Initialization */
4:  send (tick 0), round_msg_0 to all;
5:
6:  if received (tick ℓ) from at least f + 1 distinct
    processes, with ℓ > k then
7:      for i := k + 1 to ℓ do
8:          k := i;
9:          action();
10: fi
11:
12: if received (tick k) from at least n − f distinct
    processes then
13:     k := k + 1;
14:     action();
15: fi
16:
17: procedure action();
18: begin
19:     if k = (r + 1)(r + 2)/2 then
20:         execute lock-step round r's step;
21:         send (tick k), round_msg_{r+1} to all processes;
22:         r := r + 1;
23:     else
24:         send (tick k) to all;
25:     fi
26: end;
```

Fig. 2. Lock-step round simulation for unknown Θ

4.2 ?Θ-Model

We will now consider the variant of the Θ-Model, where some unknown Θ holds always (and thus messages are never lost). We show that eventual lock-step rounds can be reached under this weak model as well. The algorithm is given in Figure 2 and is very similar to the algorithm given in Figure 1. Since Θ is unknown, the algorithm operates by continuously increasing the number of "micro ticks" k between two "macro ticks" r by one.

The algorithms of Figure 1 and Figure 2 differ only in `line 19` in the function *action()*. Lemma 3 and Theorem 1 — and hence Lemma 6 — hence apply to this section's algorithm as well. The following Lemma 7 is a trivial consequence of continuously increasing the round durations.

Lemma 7. *For every $r \geq -1$, let $k(r)$ be such that $\varrho_p^r = \sigma_p^{k(r)}$, for any correct process p. Then, there exists a round s such that $k(r+1) - k(r) \geq \Xi \geq 3\Theta$ for all rounds $r \geq s$.*

Proof. From Figure 2, it is apparent that $k(r) = (r+1)(r+2)/2 - 1$. Hence, $k(r+1) - k(r) = r + 2$, which is monotonically increasing in r and hence grows without bound. □

Consequently, from round $S \geq s$ with $k(S) > k_{\mathrm{GST}}$ on, (the proof of) Theorem 2 applies. We thus obtain:

Theorem 3. *For $\Xi \geq 3\Theta$, the algorithm given in Figure 2 is a correct eventual lock-step round simulation.*

4.3 $\diamond?\Theta$-Model

In this variant of the Θ-Model, the unknown Θ holds only after some unknown GST. The algorithm given in Figure 2 also guarantees eventual lock-step rounds in this model. In fact, the proofs in Section 4.2 are not invalidated by the fact that Θ holds only eventually.

4.4 Weaker Variants of the Θ-Model

The algorithm given in Figure 2 increases the duration of macro ticks, as measured in micro ticks, in every round. Thus, for any constant Θ that eventually holds, eventually every round is sufficiently long to ensure lock-step rounds. Since the actual increase of the round duration is algorithm-dependent, however, it is possible to devise algorithms where the round duration increases in a particular way. Therefore, the "constant Θ" assumption could be weakened, i.e., substituted with an assumption that some $\Theta(t)$ eventually increases at most according to some function (e.g. constant, linear, exponential). For any such assumption, it is possible to devise an algorithm for which the round durations increases faster than $\Theta(t)$. Consequently, eventual lock-step rounds can also be achieved in such very weak variants of the Θ-Model.

5 Applicability Issues

In this section, we will discuss some general aspects of the applicability of the Θ-Model. As already mentioned in Section 1, relaxing some assumptions (synchrony in our case) in a model is usually beneficial in terms of coverage, but comes at a price (continously sending messages in our case).

The FLP impossibility [21] of consensus in asynchronous systems serves as a good example of such a tradeoff: It is well-known that consensus is easily solvable, even in the presence of Byzantine failures, in synchronous systems [30]. Sacrificing the synchrony assumption comes at a very high price, namely, an impossibility result: Consensus cannot be solved in asynchronous systems, even if we allow at most a single crash failure [21]. Despite of this fact, however, a huge amount of research on consensus in asynchronous systems has been — and is being — conducted, which established several ways to circumvent this impossibility, e.g., by means of failure detectors [11] or partial synchrony [13, 15].

With respect to the usage in real systems, it very much depends on the particular application whether relaxed synchrony conditions are really worth the price (impossibilities, reduced coverage and/or performance, etc.). There is no such thing as a "best" computing model in general — one has to carefully analyze and balance the strengths and weaknesses of every model in order to identify the most suitable model for a given application. In the following Section 5.1, we will outline some applications where the Θ-Model is useful; Section 5.2 is devoted to its limitations and some ways to alleviate those.

5.1 Sample applications

Θ-algorithms are easily portable, since there is no need to compile explicit worst-case timing bounds into those — some bound Θ on the worst-case end-to-end delay ratio is sufficient. We have strong evidence that Θ is less susceptible to changes in the implementation technology of the underlying system than the former in certain applications. This is particularly true in case of VLSI circuits, which are a promising application domain for fault-tolerant distributed algorithms in general. Due to continuously shrinking feature sizes and increasing clock speeds, today's deep sub-micron VLSI chips can no longer be viewed as monolithic blocks of synchronous hardware, but must rather be considered as loosely-coupled distributed systems: Every chip consists of a huge number of asynchronous logic gates, which concurrently and *continuously* compute the value of their output signals based on the value of their input signals. Their behavior can hence be viewed as some message-driven processing of asynchronous messages, represented by input signal transitions [16,46], with the nice property that continuously sending messages does not increase the workload of a gate.

In the context of our DARTS project[6], we adapted the core of the algorithm given in Figure 1 for fault-tolerant distributed clock generation in VLSI Systems-on-Chip (SoC), see [22] for details.[7] In the VLSI implementation of this algorithm [18],[8] the actual value of Θ appears only in the number of stages of certain FIFO buffers ("elastic pipelines"). Easy portability of Θ-algorithms was confirmed by the fact that there was no need to change the algorithm at all when making the transition from our prototype FPGA implementation to a space-hardened ASIC, for example. This "invariance" of Θ follows primarily from the facts that (1)

[6] Supported by the Austrian bm:vit FIT-IT project *DARTS* (project number 809456-SCK/SAI); for details see *http://ti.tuwien.ac.at/ecs/research/projects/darts/*.

[7] An international patent [43] is pending.

[8] This implementation shows that the unbounded memory requirement of the algorithm in Figure 1 can be avoided.

using a faster implementation technology affects minimum and maximum delay paths in the same way, and (2) one has some control on the routing of the interconnecting wires — and hence of the relative delays — during place-and-route in VLSI design.

Our DARTS clocks also provide a good example why it may be advantageous, even vital, for an algorithm not to depend on bounded-drift clocks. Unlike traditional clock synchronization algorithms, which synchronize free-running clocks driven by a local oscillator, our DARTS clocks do not need any oscillator. Rather, the clock signals just emerge from the interaction of the simple local clock generation algorithms. Obviously, such a solution is only possible if the clock generation algorithms itself are clockless.[9] Note that this feature makes DARTS clocks particularly attractive for systems operated in harsh environments (e.g., aerospace applications), where quartz oscillators sometimes cause problems — in particular, could fail to start oscillating upon power up.

Apart from the very specific VLSI context, we observed that Θ-algorithms perform very well also in some more classic distributed applications, as argued in [31, 51]: Both a worst case schedulability analysis of a distributed system based on Deterministic Ethernet [25] and experiments in a workstation network using switched Ethernet technology [5] revealed that the Θ-Model has indeed a good coverage in those particular applications. It goes without saying, however, that these examples cannot be considered representative for classic distributed applications in general.

5.2 Muteness and Performance Issues

Θ-algorithms like the lock-step round simulation in Figure 1 are solely message-driven, which implies that some messages must be in transit at any time. As a consequence, one may be concerned about (1) the potential of the system to turn mute in case of total message loss, and (2) the system load generated by Θ-algorithms.

As far as potential muteness is concerned, we first note that even the Θ-algorithms analyzed in this paper allow considerable message loss: Reliable links were only required for messages exchanged by *correct* processes, which means that lost messages can be attributed to faulty processes. Moreover, [52] shows that our algorithms can be generalized to work under our perception-based hybrid failure model [42, 44], which allows a considerable number of messages exchanged by correct processes to get lost as well (provided that n is chosen sufficiently large). This means that only massive message loss could turn a Θ-algorithm mute, which is an event that should have a very small probability in real systems anyway. If this probability was unacceptably high, one could augment the system [and the Θ-Model] by some "deadlock prevention event", as in done in [29] in the context of self-stabilizing systems [14]. In view of all these possibilities to deal with message loss, we do not think that potential muteness is a real problem.

Concerning the load generated by Θ-algorithms, we first note that there are applications where continuously sending messages comes for free. Apart from the VLSI context described earlier, this may also be the case for low-level "network-centric" applications: Links in modern high-speed networks, like SpaceWire [39], continuously exchange heartbeat data for maintaining communication synchrony and failure detection when there is no real data to be sent. Continuously sending messages would not increase the load here, but only reduce the amount of (useless) heartbeat data sent.

Moreover, for distributed systems based on networks with some reasonably large bandwidth×delay product, the overhead caused by Θ-algorithms is typically small: Consider the extreme case of a satellite broadcast communication link, for example, where the end-to-end communication delay typically is in the order of 300 ms. With up to $n = 10$ processors, 1,000 bit long messages, and a link throughput of 2 megabit/second, the link occupancy time for this paper's algorithm would be 5 ms per round, entailing a communication overhead smaller than 2 percent (the round duration is 300 ms at least). Given that the bandwidth provided by modern computer networks has tremendously increased due to the advances in communications technology, while the spatial distribution and hence the delays remained the same, the delay×bandwidth product shows a clear rising trend. If Θ-algorithms are used for reasonably low-level (i.e., close to the network) distributed applications, the system load could hence be acceptable.

In systems where this is not the case, there is a way to reduce the load generated by Θ-algorithms — without affecting correctness. Note, however, that this extension is not fully compatible with the purely message-driven model introduced in Section 2. The idea, introduced in [31], is to use a local "delay timer" for introducing pauses in between consecutive ticks. Those "delay timers" do not need to satisfy bounded drift conditions and could hence be implemented without hardware clocks (e.g. by counting suitable computing events).

We will explain the method by adapting the lock-step round simulation of Figure 1: The modified algorithm is shown in Figure 3. It assumes that there is a function $delay_D$, which just spin-loops for some time D satisfying $D^- \leq D \leq D^+$, where $D^- \geq 0$ and, in particular, $D^+ < \infty$ can be unknown. This function is called before (*tick* $k+1$) is sent, in the case where clock progress is due to the advance rule `line 12`. Note that there is no need for a delay if clock progress is due to the catch-up rule `line 6`: Recalling Lemma 3, it is apparent that it

[9] An example of a more classic distributed system where clockless algorithms are considered useful are *wireless sensor networks* (WSNs) [6, 17]: Sensor network nodes are typically battery-powered, which makes low power consumption a key issue. Replacing the usual time-driven algorithms, which spend most of the time in an idle loop, waiting for incoming messages, by a message-driven algorithm clearly holds potential for energy saving. Whereas standard Θ-algorithms are certainly not suitable for WSNs, extensions similar to the one described in Section 5.2 might turn out useful here.

```
0:  VAR k : integer := 0;    // clock value
1:  VAR r : integer := 0;    // round number
2:
3:  /* Initialization */
4:  send (tick 0), round_msg_0 to all;
5:
6:  if received (tick ℓ) from at least f + 1 distinct
    processes, with ℓ > k then
7:      for i := k + 1 to ℓ do
8:          k := i;
9:          action(0);
10: fi
11:
12: if received (tick k) from at least n − f distinct
    processes then
13:     k := k + 1;
14:     action(D);
15: fi
16:
17: procedure action(d);
18: begin
19:     if k = (r + 1)Ξ then
20:         execute lock-step round r's step;
21:         r := r + 1;
22:         if d > 0 then delay_D fi ;
23:         send (tick k), round_msg_{r+1} to all processes;
24:     else
25:         if d > 0 then delay_D fi ;
26:         send (tick k) to all;
27:     fi
28: end;
```

Fig. 3. Lock-step round simulation for known Θ, with local delay for reducing system load

suffices to slow down the progress of $C_{\max}(t)$ in order to slow down the overall system.

Let $\hat{\sigma}_p^k$ denote the time when (*tick* k) is actually sent (after its preceding $delay_D$, if any), and σ_p^k be the time when p sets its clock to $k+1$ and calls the $delay_D$, if any, that precedes the sending of (*tick* $k+1$). Similarly, let $\hat{\varrho}_p^r$ denote the time when p sends its round r message, and ϱ_p^r be the time when p executes its round r computing step (and switches to round $r+1$). Note that obviously $D^- \leq \hat{\sigma}_p^{k+1} - \sigma_p^k \leq D^+$ if $C_p(t)$ made progress via `line 12`, whereas $\hat{\sigma}_p^{k+1} = \sigma_p^k$ otherwise.

The following lemma follows immediately from the code of Figure 3:

Lemma 8. *For every correct process p and every $r \geq -1$, it holds that $\varrho_p^r = \sigma_p^k$ and $\hat{\varrho}_p^{r+1} = \hat{\sigma}_p^{k+1}$ for $k = (r+1)\Xi - 1$.*

For our analysis, we first note that the modification of the algorithm relative to Figure 1 does not essentially affect the results of Theorem 1. More specifically, using exactly the same proof, Theorem 4 given below can be established. Note that we only changed the preconditions in (P) and (U), where *setting clocks to k* (at time σ_p^{k-1}) has been replaced by *sending the (tick k) message* ($\hat{\sigma}_p^k$).

Theorem 4 (Clock Synchronization). *In systems of $n \geq 3f+1$ processes adhering to the perpetual Θ-Model, the algorithm given in Figure 3 achieves:*

(P) Progress. *If all correct processes send (tick k) by time t, then every correct process sets its clock at least to $k+1$ by time $t + \tau^+$.*

(U) Unforgeability. *If no correct process sends (tick k) by time t, then no correct process sets its clock to $k+1$ by time $t + \tau^-$ or earlier.*

(Q) Quasi Simultaneity. *If some correct process sets its clock at least to $k > 0$ at time t, then every correct process sets its clock at least to $k-1$ by time $t + \varepsilon$.*

(S) Simultaneity. *If some correct process sets its clock at least to k at time t, then every correct process sets its clock at least to k by time $t + \tau^+ + \varepsilon$.*

In addition, Lemma 5 continues to hold when (P) and (U) in Theorem 1 is replaced by the variants given in Theorem 4. Hence, we just need a generalization of Lemma 6 that distinguishes $\hat{\sigma}_p^{k+1}$ and σ_p^k thus takes into account $delay_D$:

Lemma 9 (Delayed Progress). *For every $R \geq P$, $P > k_{\mathrm{GST}}$ and any two correct processes p and q,*

$$|\sigma_p^R - \sigma_q^R| \leq \tau^+ + \varepsilon \tag{5}$$

$$\sigma_p^R - \hat{\sigma}_q^P \geq (R-P)(\tau^- + D^-) + \tau^- - \tau^+ - \varepsilon \tag{6}$$

$$\sigma_p^R - \hat{\sigma}_q^P \leq (R-P)(\tau^+ + D^+) + \tau^+ + \tau^+ + \varepsilon. \tag{7}$$

Proof. Equation (5) follows immediately from the simultaneity property (S). In addition, (6) and (7) for $P = R$ follow directly from (3) and (4) in Lemma 6 for $R - P = 1$, since $\hat{\sigma}_q^P$ here is just the instant of sending (*tick* P), which takes place at σ_q^{P-1} there.

We can hence assume that (6) holds for $k - 1 = R - P - 1 \geq 0$ and have to show that it holds for $k = R - P$ as well. Since process p sets its clock to $R+1$ at time σ_p^R, at least one correct process p_1 must have called $delay_D$ — preceding the sending of (*tick* $R+1$) — by time $\sigma_p^R - \tau^- - D^-$ at latest. This follows from the unforgeability property (U) in Theorem 4 in conjunction with Lemma 3, which asserts that there is a correct process p_1 that has advanced its clock via `line 12` and hence has called $delay_D$ before sending its (*tick* $R+1$). Hence $\sigma_p^R - \sigma_{p_1}^{R-1} \geq \tau^- + D^-$. By the induction hypothesis, we have $\sigma_{p_1}^{R-1} - \hat{\sigma}_q^P \geq (R-P-1)(\tau^- + D^-) + \tau^- - \tau^+ - \varepsilon$, from which (6) follows immediately.

For the upper bound, we can assume that (7) holds for $k - 1 = R - P - 1 \geq 0$ and have to show that it holds for $k = R - P$ as well. Since we know that even the last correct process sets its clock to R at some time $\sigma_{\max}^{R-1}$ satisfying $\sigma_{\max}^{R-1} - \hat{\sigma}_q^P \leq (R-P-1)(\tau^+ + D^+) + \tau^+ + \tau^+ + \varepsilon$ by the induction hypothesis, it follows from the progress property (P) in Theorem 4 and the maximum delay $D \leq D^+$ of $delay_D$ that process p sets its clock to $R+1$ by time $\sigma_{\max}^{R-1} + \tau^+ + D^+$, from where (7) follows. □

We hence obtain the following counterpart of Theorem 2:

Theorem 5. *Let Ξ be an integer with $\Xi - 1 > \lceil (3\Theta - 2) \cdot R' \rceil$ with $R' \geq R = \tau^-/(\tau^- + D^-)$, where R is monotonically increasing with respect to τ^- and $0 \leq R \leq \min\{\tau^-/D^-, 1\}$. From $k_{\mathrm{GST}} + 1$ on, the algorithm given in Figure 3 is a correct lock-step round simulation.*

Proof. Let $k > k_{\mathrm{GST}} + \Xi$ be any integer such that $k = (r+1)\Xi - 1$ for some integer $r \geq 0$. We know that every correct process p sends its lock-step round $r+1$ message along with $(tick\ k+1)$, and its lock-step round r message along with $(tick\ k+1-\Xi)$. From Lemma 8, it hence follows that $\hat{\sigma}_p^{k+1-\Xi}$ is the time when the lock-step round r message is sent, whereas σ_p^k is the time when the lock-step round r computation step is executed.

The lock-step round simulation is correct if $\sigma_{first}^k - \hat{\sigma}_q^{k+1-\Xi} - \tau^+ > 0$ for every correct process q. Using Lemma 9 and

$$\Xi - 1 > \frac{(3\Theta - 2)\tau^-}{\tau^- + D^-} \geq \frac{3\tau^+ - 2\tau^-}{\tau^- + D^-},$$

we easily obtain

$$\begin{aligned}\sigma_{first}^k - \hat{\sigma}_q^{k+1-\Xi} - \tau^+ &\geq (\Xi - 1)(\tau^- + D^-) + \\ &\quad + \tau^- - 2\tau^+ - \varepsilon \\ &> 3\tau^+ - 2\tau^- - 3\tau^+ + 2\tau^- = 0\end{aligned}$$

as required.

The "reduction factor" $R = \tau^-/(\tau^- + D^-)$ is monotonically increasing in τ^- and obviously satisfies

$$0 \leq \frac{\tau^-}{\tau^- + D^-} \leq \min\left\{\frac{\tau^-}{\tau^-}, \frac{\tau^-}{D^-}\right\} = \min\left\{\tau^-/D^-, 1\right\}.$$

This confirms the bound for R given in our theorem. □

Clearly, by increasing D^-, the message load is reduced since the progress of the entire system is slowed down. If one chooses $R' = 1$, i.e., $\Xi - 1 > 3\Theta - 2$, which is legitimate since $R \leq 1$ according to the lower bound, Theorem 5 reveals that the lock-step round simulation works even if D^- is arbitrary, that is, its correctness does not depend on $delay_D$. On the other hand, if a bound D^- can be guaranteed, it also follows from Theorem 5 that Ξ could be reduced to the minimum value $\Xi = 2$ by choosing D^- to be sufficiently large.

6 Relation to Other Models

Several different partially synchronous computing models [2, 8, 11, 13, 15, 20, 41, 47, 48] have been proposed in the literature, which differ in the synchrony assumptions added to the asynchronous FLP model [21]. After a brief description of the core features of those models, we will compare the Θ-Models with the classic partially synchronous models of [15] in some detail in Section 6.1, and provide a brief relation to the other existing partially synchronous models in Section 6.2.

The well-known synchronous model may be viewed as the FLP model augmented by a priori bounds on real-time transmission delays Δ and computing step times σ, as well as bounded drift local clocks. The same assumptions are used in the timed asynchronous model [12]; however, Δ and σ may be violated arbitrarily during "unstable periods" here. The approach described in [47] requires only a small part of the system, the timely computing base, to obey Δ and σ; the remaining payload part of the system can be asynchronous. A different "system partitioning" underlies the fast failure detector approach advocated in [24], where only failure detector level messages and computations must be timely.

Weaker properties are added to the FLP model in the classic partially synchronous [11, 15] and semi-synchronous [8, 41] computing models: Instead of the computing step time σ, they only incorporate a bound Φ on the relative speed of processes, in addition to the standard synchronous transmission delay bound Δ. All those models allow a process to time-out messages via some local notion of real-time: The semi-synchronous models [8, 41] assume that local real-time clocks are available. The models of [15] use the computing step time of the fastest process as the units of "real-time"; hence, receive steps in a spin-loop with loop-count Δ is sufficient for timing out the maximum message delay here. An even older partially synchronous model is the Archimedean model introduced in [48]. It adds a bounded ratio $s \geq u/c$ on the minimum computing step time c and the maximum computing step time + transmission delay u to the FLP model. Again, any process can timeout a message by means of a local spin-loop with loop count s here.

An even more relaxed way of adding synchrony properties to asynchronous systems underlies the chase for the weakest system model for implementing the Ω failure detector [11] in systems with crash failures [1, 3, 4, 7, 26, 27, 34, 37]; an extension to Byzantine failures can be found in [2]. These Weak Timely Link (WTL) models can be viewed as a "spatial" relaxation of the classic partially synchronous or semi-synchronous models: The currently weakest of these models [26, 27] requires just a single process p (a timely f-source) with f eventual timely outgoing links, connecting p to a time-variant set of f receiver processes. Note that communication over all the other links in the system can be totally asynchronous, i.e., need not satisfy any time bound.

Another model that is close to pure asynchrony is the Finite Average Response time (FAR) model introduced in [19, 20]. The properties added to the FLP model are an unknown lower bound for the computing step time, and an unknown finite average of the round-trip delays between any pair of correct processes. The latter allows round-trip delays to be increasing without bound, provided that there are sufficiently many short round-trips in between that compensate the resulting increase of the average. Due to the computing step time lower bound, any process can implement a weak bounded drift clock via a local spin-loop here, which allows to safely timeout messages by using timeout values learned at runtime.

6.1 Relation to Classic Partially Synchronous Models

In this section, we will show in detail how our Θ-Models and the classic partially synchronous models [11, 15] are related to each other. We will look at several orthogonal aspects of computing models separately, namely, (1) failure model, (2) added timing constraints, (3) solvability of problems, and (4) self-stabilization. It will turn out that the Θ-Models are weaker than their classic partial synchronous counterparts in most aspects, see Table 1. We will use the abbreviation $A \prec B$ (resp. $A \preceq B$) to denote the fact that model A is strictly weaker than B (resp. strictly weaker or equivalent) with respect to some aspect that will be clear from the context. We say that two models are equivalent, denoted $A \approx B$, if both $A \preceq B$ and $B \preceq A$.

The classic partially synchronous models [11, 15] (referred to as *PS*-Models here for brevity) extend the FLP model by adding an absolute upper bound Δ on message transmission delays (not end-to-end) and an upper bound Φ on the relative computing speed of any two processes. In the original paper, a single computing step can either be a (unicast) send step or a receive step, but not both. The bound Δ specifies some real-time duration, which ensures that every message sent at real-time $t \geq$ GST is received at the latest at the first reception step of the receiver at some time $t' \geq t + \Delta$. Similarly, it is assumed that every process takes at least 1 and at most Φ computing steps during any time interval of duration Φ.

When taking a closer look at the definition of Φ and how it is used in [15], one finds that the unit of "real-time" is actually the computing step time $c > 0$ of the fastest process in the system. The latter forms a discrete "real-time" base, to which all processes in the system are synchronized: Every process can make at most one step per tick, and must at least make one step within Φ ticks. Hence, obviously, no computing step can be taken in zero time. Similarly, Δ gives the number of "real-time" ticks that occur while a message is in transit on some link. *PS*-algorithms essentially exploit the fact that Δ computing steps are sufficient for timing out any message, where the sequence of the steps is determined by the algorithm.

The bounds Δ and Φ need not necessarily be known a priori to the processes: Φ and Δ in [15] can either be unknown, or known but hold only after some unknown global stabilization time GST. Again, known Δ means that Δ holds in every run, while unknown Δ means that in every run β, there is some Δ that holds in β [15].

A generalized partially synchronous model integrating those two models was introduced in [11]: It assumes that relative speeds, delays and message losses are arbitrary up to GST; after GST, no message may be lost between correct processes and all relative speeds and all communication delays must obey the (possibly) unknown upper bounds Φ and Δ, respectively. To facilitate comparison with Definitions 2 and 3, we introduce the following Definition 8.

Definition 8 (Partially Synchronous Models). *The (partially) synchronous models are denoted as follows:*

- *PS-Model: Δ and Φ are known and hold perpetually (this constitutes the partially synchronous model from [15] that is closest to the synchronous model)*
- *?PS-Model: Δ and Φ are unknown and hold perpetually (one of the models in [15])*
- *$\diamond$PS-Model: Δ and Φ are known and hold eventually (one of the models in [15])*
- *$\diamond$?PS-Model: Δ and Φ are unknown and hold eventually (the generalized partially synchronous model from [11])*

Adding the reliable links assumption — i.e., no message loss between correct processes before GST *(where applicable) — yields the following models:*

- *PS_r-Model: PS-Model*
- *?PS_r-Model: ?PS-Model*
- *$\diamond PS_r$-Model: $\diamond$PS-Model without message loss before* GST
- *$\diamond$?PS_r-Model: $\diamond$?PS-Model without message loss before* GST*, i.e., the ?PS-Model (since the unknown delay may be chosen sufficiently large to also cover the delays of all messages sent before* GST*)*

Comparison with Definition 2 reveals that every Θ-Model has a corresponding *PS*-Model (with and without message loss before GST). For example, $\diamond?\Theta$ corresponds to $\diamond?PS$ and $\diamond?PS_r$.

Failure Model

Both *PS*-Models and Θ-Models allow the implementation of (eventual) lock-step rounds in the presence of Byzantine failures. Hence, those models can be considered equivalent with respect to the process failure model. However, $\diamond PS$ and $\diamond?PS$ allow message loss (link failures) before GST even between correct processes, whereas the corresponding Θ-Models $\diamond\Theta$ and $\diamond?\Theta$ (as well as $\diamond PS_r$ and $\diamond?PS_r$) assume that even messages sent before GST from correct processes to correct processes are eventually delivered.

Consequently, as summarized in Relation 1 below, some of the *PS*-Models ($\diamond PS \prec \diamond\Theta$ and $\diamond?PS \prec \diamond?\Theta$) are weaker than the corresponding Θ-Models. We note, however, that this is not a consequence of the Θ-Model's timing assumptions, but rather a consequence of the message-driven execution model: Θ-algorithms may turn mute if unrestricted message loss is allowed.

Relation 1 (Failures) *Regarding the communication failure model — message loss before* GST *— we have*

- $PS \approx PS_r \approx \Theta$
- $?PS \approx ?PS_r \approx ?\Theta$
- $\diamond PS \prec \diamond PS_r \approx \diamond\Theta$
- $\diamond?PS \prec \diamond?PS_r \approx \diamond?\Theta$

We also note that it is possible to *simulate* $\diamond PS_r$ atop of $\diamond PS$, as well as $\diamond?PS_r$ atop of $\diamond?PS$: Since only a finite number of messages can be lost before GST, it suffices to use a "full message history" communication protocol (with the possible improvement of discarding previous

messages as soon as they have been acknowledged) to ensure that every message sent before GST will eventually be received. Although this simulation cannot completely mask the effect of message loss before GST, due to the fact that the message size is increasing, it nevertheless shows that even total message loss is not a problem in the *PS*-Model. By contrast, using an analogous simulation in the message-driven Θ-Model could only be used to overcome partial message loss between correct processes; a total message loss would again turn the whole system mute.

Added timing properties

One can also compare Θ-Models and *PS*-Models via the timing constraints added to the asynchronous model. To this end, we will investigate the end-to-end delays in executions of partially synchronous algorithms. To facilitate a fair comparison with Θ-algorithms, we must exclude algorithms that deliberately produce unbounded end-to-end delays, simply by not executing receive steps: For example, an algorithm that executes an ever increasing number of send steps in between any two receive steps inevitably suffers from unbounded end-to-end delays. Such "non-receptive" algorithms are obviously not particularly useful. We will hence restrict our attention to *receptive PS*-algorithms, which take only a bounded number of send steps in between two receive steps.

Our first Lemma 10 will reveal that every run of a receptive PS_r-algorithm satisfies the assumptions of the corresponding Θ-Model.

Lemma 10 (Θ in *PS* Systems). *Every run of a receptive algorithm in one of the PS_r-Models from Definition 8 also satisfies the assumptions of the corresponding static Θ-Model from Definition 2 (and hence also Definition 3) for some suitable Θ.*

Proof. Suppose we are given a system that satisfies a PS_r-Model with (possibly unknown) Δ and Φ from some time GST on, depending on which of the models PS_r, $?PS_r$, $\diamond PS_r$, $\diamond ?PS_r$ applies. Let R denote the (possibly unknown) bound on the number of send steps in between any two receive steps of the receptive algorithm.

Since the system model of [15] does not allow a single computing step to both receive and send some message(s) simultaneously, any end-to-end delay involves at least one step time. For the minimum end-to-end delay, the best case is just a single send step (of duration $c > 0$) after the last receive step, and a message that travels in zero time and arrives at the receiver exactly when it is about to execute its receive step. Hence, $\tau^- \geq c > 0$, cf. Definition 1. For the maximum end-to-end delay, we have R send steps of duration Φc after the sender's last receive step, and a message that takes Δc to arrive at the receiver slightly after it executed its receive step. Hence, it may take up to $(R+1)\Phi c$ for the receiver to actually process this message. Consequently, $\tau^+ \leq ((2R+1)\Phi+\Delta)c$.

Choosing $\Theta = (2R+1)\Phi + \Delta$ in Definition 2 hence ensures that all end-to-end delays satisfy the delay ratio Θ as asserted. Clearly, Θ is known iff Φ and Δ are known, and holds from GST on. Hence, the system also satisfies the corresponding Θ-Model from Definition 2. □

On the other hand, there is no analog to Lemma 10 in the other direction. Informally, this is because the *PS*-Model requires some local synchrony conditions (bounded ratio of computing steps) and some communication synchrony conditions (bounded transmission delays) to hold *independently* of each other. Whereas independent computation and communication synchrony also implies end-to-end synchrony (i.e., of end-to-end computation and communication), the converse is not necessarily true: There can be bounded synchrony with respect to end-to-end delays without individually bounded computation and communication synchrony, as Lemma 11 proves:

Lemma 11. *Given some end-to-end delay ratio $\Theta > 1$ and any receptive algorithm $\mathcal{A}$ designed for one of the PS_r-Models of Definition 8, there is an execution E^Θ of $\mathcal{A}$ that satisfies Θ but ist not admissible in the corresponding PS_r-Model.*

Proof. Fix $\Theta > 1$, $c > 0$ and any receptive algorithm $\mathcal{A}$ with at most $R \geq 1$ steps in between any two receive steps. We will first construct a sequence of executions $E(\Phi_k, \Delta_k)$ with $\lim_{k\to\infty} \Phi_k = \infty$ and $\lim_{k\to\infty} \Delta_k = \infty$, with the property that for finite Φ_k and Δ_k all end-to-end delays in $E(\Phi_k, \Delta_k)$ satisfy Θ.

Let Φ', Δ' be some arbitrary choice of the parameters. We arbitrarily divide the set of processes into two non-empty sets, the *slow* and the *fast* processes, respectively. Consider the following admissible execution $E(\Phi', \Delta')$ of $\mathcal{A}$: The fast processes take steps in perfect synchrony every c time-units, i.e., with maximum speed, whereas the slow processes executes their steps every $\Phi' c$ time-units, i.e., with minimal speed. Moreover, all messages sent take the transmission delay Δ' (i.e., a message sent at time t is received at the first receive step of the receiver after $t + \Delta' c$).

The end-to-end delay of a message m is measured from the last receive step of the sender before it sends m to the receive step where m is delivered. There are only four cases for the end-to-end delays in $E(\Phi', \Delta')$:

- All messages exchanged between fast processes have an end-to-end delay between $(\Delta' + 1)c = \tau^-$ and $(\Delta' + 2R + 1)c$.
- All messages sent from fast processes to slow ones have an end-to-end delay between $(\Delta' + 1)c$ (when the message arrives at the time of the a receive step of the slow process) and $(\Delta' + R + (R+1)\Phi')c$ (when the message arrives immediately after a receive step of the slow process).
- Similarly, all messages from slow processes to fast ones have an end-to-end delay between $(\Phi' + \Delta')c$ (when the message arrives at the time of a receive step of a fast process) and $(R\Phi' + \Delta' + R + 1)c$.
- End-to-end delays between two slow processes are between $(\Phi' + \Delta')c$ and $((2R+1)\Phi' + \Delta')c = \tau^+$.

Consequently, we have

$$\frac{\tau^+}{\tau^-} = \frac{(2R+1)\Phi' + \Delta'}{\Delta' + 1}. \tag{8}$$

Given Θ, one can obviously determine some Φ' and Δ' such that

$$\frac{(2R+1)\Phi' + \Delta' + 1}{\Delta' + 1} \leq \Theta,$$

by just choosing Δ' sufficiently large. Θ is thus a bound on the end-to-end delay ratio in $E(\Phi', \Delta')$.

For every $k \geq 0$, let

$$\Phi_k = \Phi' + \left\lfloor \frac{k \cdot \Phi'}{\Delta' + 1} \right\rfloor \text{ and } \Delta_k = \Delta' + k,$$

which ensures $\Phi_k \to \infty$ and $\Delta_k \to \infty$. The resulting executions $E(\Phi_k, \Delta_k)$ satisfy upper and lower bounds on end-to-end delays τ_k^+ and τ_k^- with

$$\begin{aligned} \frac{\tau_k^+}{\tau_k^-} &= \frac{(2R+1)\Phi_k + \Delta_k}{\Delta_k + 1} \\ &< \frac{(2R+1)\Phi_k + \Delta_k + 1}{\Delta_k + 1} = \Theta_k. \end{aligned} \tag{9}$$

We derive

$$\begin{aligned} \Theta_k &\leq \frac{(2R+1)\Phi' + k \cdot \frac{(2R+1)\Phi'}{\Delta'+1} + \Delta' + k + 1}{\Delta' + k + 1} \\ &= \frac{(2R+1)\Phi' + \Delta' + 1}{\Delta' + k + 1} + \\ &\quad + \frac{\frac{k}{\Delta'+1} \cdot \big((2R+1)\Phi' + \Delta' + 1\big)}{\Delta' + k + 1} \\ &= \frac{(2R+1)\Phi' + \Delta' + 1}{\Delta' + 1} \cdot \frac{1 + \frac{k}{\Delta'+1}}{1 + \frac{k}{\Delta'+1}} \leq \Theta. \end{aligned} \tag{10}$$

From (9) and (10), it follows that if $E(\Phi', \Delta')$ satisfies Θ, then the execution $E(\Phi_k, \Delta_k)$ satisfies Θ for all $k \geq 0$.

To prove our lemma in case of the PS_r-Model with any known Φ and Δ, we just choose some index ℓ such that $\Phi_\ell \geq \Phi$ and $\Delta_\ell \geq \Delta$ and set $E^\Theta = E(\Phi_\ell, \Delta_\ell)$. Since E^Θ respects Θ but violates Φ and Δ, we have the required contradiction.

In the case of the remaining PS_r-Models $?PS_r$ and $\diamond?PS_r$, we inductively construct a single exection E^Θ out of (pieces of) $E(\Phi_k, \Delta_k)$, $k \geq 0$: We start with an arbitrary prefix of $E(\Phi_0, \Delta_0)$, denoted E_0^Θ. Now assume that we have already constructed E^Θ up to E_k^Θ, for $k \geq 0$. To extend E_k^Θ to E_{k+1}^Θ, we first increase the delays of all messages sent after E_k^Θ from Δ_k to Δ_{k+1}. The end-to-end delays during this "transition phase" are between $\hat{\tau}_k^+ = ((2R+1)\Phi_k + \Delta_{k+1}) \cdot c$ and $\hat{\tau}_k^- = \tau_k^- = (\Delta_k + 1) \cdot c$. Since

$$\begin{aligned} \frac{\hat{\tau}_k^+}{\hat{\tau}_k^-} &= \frac{(2R+1)\Phi_k + \Delta_{k+1}}{\Delta_k + 1} \\ &= \frac{(2R+1)\Phi_k + \Delta_k + 1}{\Delta_k + 1} = \Theta_k \leq \Theta \end{aligned} \tag{11}$$

according to (10), the end-to-end delay ratio Θ holds. Finally, when the last message sent throughout E_k^Θ is eventually delivered, we also increase the step time of the slow processes from Φ_k to Φ_{k+1} and hence proceed, for an arbitrary number of steps, according to $E(\Phi_{k+1}, \Delta_{k+1})$. This completes the construction of E_{k+1}^Θ.

In the so constructed execution E^Θ, the end-to-end delays of all messages in transit at the same time obey Θ, but there are no finite bounds on transmission delays and relative processing speeds in E^Θ. This completes the proof of Lemma 11. □

Consequently, we obtain:

Relation 2 (Timing) *With respect to the timing properties added to the asynchronous model, we have*

- $\Theta \prec PS_r \approx PS$
- $?\Theta \prec ?PS_r \approx ?PS$
- $\diamond\Theta \prec \diamond PS_r \approx \diamond PS$
- $\diamond?\Theta \prec \diamond?PS_r \approx \diamond?PS$

The dynamic Θ-Model allows us to develop some additional arguments, which support this claim also with respect to model coverage.

Our formulation of the *PS*-Model reveals that the computing step time c, which relates the model's "real-time" to real-time, is in fact arbitrary. This is very much like in the Θ-Model, where the minimal end-to-end delay τ^- is also arbitrary. We could hence even define something like a "dynamic *PS*-Model", by letting $c(t)$ vary with real-time t, and employ a timing transformation technique as in Section 3. However, there is a subtle difference: The *PS*-Model constrains the transmission delay of any message to at most $c\Delta$ in real-time, which involves c and hence implies a strong correlation between communication delay and computing step time. Since network transmission and local computation are in reality independent of each other, however, such a correlation is difficult to justify in practice. By contrast, it is possible to justify such a correlation between the maximum and minimum end-to-end delay [5,25,31,51] in the Θ-Model.

It follows that the Θ-Model allows runs with ever increasing end-to-end delays (a similarity to the FAR-Model [20]). By contrast, in the *PS*-Model, delays have to stop increasing eventually in order to ensure that the bound Δ on the transmission delays is not violated.

Regarding the ability to describe real systems, the *PS*-Model has another — somewhat related — peculiarity that is avoided by the Θ-Model: Even the slowest process p must be able to receive *all* the messages that were sent to it by fast processes Δ "real-time" steps ago. Assume that some ℓ fast processes send a message to a single receiver process p at every computing step. Consequently, a fast receiver process p must be able to receive at least ℓ messages at every single step of real-time duration c. Otherwise, messages sent Δ "real-time" steps ago could not be received in time. A slow process p, on the other hand, must be able to receive $\ell\Phi$ messages in a single step of real-time duration $c\Phi$. Note that this holds true irrespectively of the value of c.

It follows that both a slow and a fast receiver p must "digest" messages at the same rate: A fast p may process some messages at every step, whereas a slow p must process many messages at once every Φ steps. This somehow contradicts the goal of modelling fast and slow processes, since the senders' computing speeds must go up

or down if the receiver's computing speed $c(t)$ goes up or down, respectively. In sharp contrast to the end-to-end delays in the Θ-Model, however, there is no obvious physical justification for such a correlation between process speeds.

Solvability aspect

The question addressed here is whether there are problems that can be solved in one of the Θ-Models but not in the corresponding *PS*-Model (or vice versa). Since we will use simulation as our primary tool here, we will restrict our attention to problems that are invariant with respect to simulation. This includes all problems that can be described via trace properties of external actions [33], for example, but excludes problems that involve e.g. performance properties (like message or time complexity) in their specification.

An important difference between Θ-Models and *PS*-Models originates in their different execution model: A computing step of a Θ-algorithm is only executed when a message has arrived (message-driven execution), whereas a computing step of a *PS*-algorithm is triggered by the passage of local time (time-driven execution). Nevertheless, Lemma 12 below shows that any message-driven Θ-algorithm can be executed in the corresponding (or a stronger) *PS*-system.

Lemma 12 (Θ-algorithms atop *PS*-systems). *Any system adhering to one of the models given in Definition 8 can run any Θ-algorithm $\mathcal{A}$ for the corresponding Θ-Model, provided that $\Theta \geq (2\alpha+1)\Phi+\Delta$ for the models where the parameters are known; α is a bound on the maximum number of messages sent by $\mathcal{A}$ in a single computing step (e.g., $\alpha = n$ if computing steps can broadcast at most one message).*

Proof. All that is needed for this simulation is to repeatedly execute a receive step in the *PS*-Model, which (1) receives messages, and (2) executes the appropriate step of the message-driven Θ-algorithm if at least one message has been received. If this results in some messages to be sent (at most α), up to α (unicast) send steps in the *PS*-Model are issued before the next receive step. In addition, if the *PS*-system adheres to $\diamond PS$ or $\diamond ?PS$, then the reliable link simulation for tolerating message loss before GST outlined after Relation 1 is employed.

Let Φ and Δ be the (possibly unknown) parameters of the underlying *PS*-system. We will determine the minimal and maximal end-to-end delay as experienced by the message-driven algorithm. We proceed exactly as in the proof of Lemma 10: For the minimum end-to-end delay, the best case is just a single send step (of duration $c > 0$) after the last receive step, and a message that travels in zero time and arrives at the receiver exactly when it is about to execute its receive step. Hence, $\tau^- \geq c > 0$, cf. Definition 1. For the maximum end-to-end delay, we have α send steps of duration Φc after the sender's last receive step, and a message that takes Δc to arrive at the receiver slightly after it executed its receive step. Hence, it may take up to $(\alpha+1)\Phi c$ for the receiver to actually process this message. Consequently, $\tau^+ \leq ((2\alpha+1)\Phi+\Delta)c$.

Choosing $\Theta = (2\alpha+1)\Phi+\Delta$ in Definition 2 hence ensures that all end-to-end delays satisfy the delay ratio Θ as asserted. Clearly, Θ is known iff Φ and Δ are known, and hold from GST on. Hence, the system also implements the corresponding Θ-Model from Definition 2. □

PS-algorithms can also be executed atop of a system that adheres to the Θ-Model, as revealed by the following Lemma 13:

Lemma 13 (*PS* algorithms atop Θ-systems). *A Θ-system adhering to one of the models given in Definition 2 can run any algorithm for the corresponding PS-model.*

Proof. Recall that our synchronizers introduced in Section 4 allow to eventually simulate lock-step rounds in any Θ-system adhering to Definition 2. Lock-step rounds are equivalent to a *PS* system with $\Phi = 1$ and $\Delta = 0$, however: The steps of the *PS* algorithm are just executed in the computing steps at the end of a round. Since all messages sent in the previous step are received before the next step, this behavior simulates $\Phi = 1$ and $\Delta = 0$ as asserted. Since executions with $\Phi = 1$ and $\Delta = 0$ are of course also admissible for general *PS* systems (with arbitrary $\Phi \geq 1$ and $\Delta \geq 0$), we are done. □

Lemma 12 in conjunction with Lemma 13 thus reveals that one can simulate any Θ-Model in the *PS*-Model and vice versa, which is summarized in Relation 3 below. We note that the above simulations differ heavily in their performance/overhead, however: Whereas the simulation of a Θ-Model in the *PS*-Model is cheap, it is costly and slow to simulate a *PS*-Model in the Θ-Model: In fact, the computing step time in the simulated algorithm is determined by the end-to-end delays — rather than the computing step times — of the underlying system.

This overhead refers to simulations — which allow to execute *any* algorithm designed for one model in the other — only, and in general allows no conclusion on how efficiently certain problems can be solved in the compared models: For example, we know that every algorithm from [15] designed for the *PS*-Model can be run atop of the corresponding Θ-Model using the simulation from Lemma 13. Doing this would be inefficient in practice, however, since the algorithms from [15] employ clock synchronization algorithms designed for the *PS*-Model for simulating eventual lock-step rounds. Since this is exactly what our synchronizers achieve, this would unnecessarily double the effort.

Relation 3 (Solvability aspect) *With respect to the solvability of problems, we have*

- $\Theta \approx PS_r \approx PS$
- $?\Theta \approx ?PS_r \approx ?PS$
- $\diamond\Theta \approx \diamond PS_r \approx \diamond PS$
- $\diamond?\Theta \approx \diamond?PS_r \approx \diamond?PS$

Note that if we allowed message loss before GST in the Θ-Model, Lemma 13 would not hold any more: In sharp contrast to time-driven algorithms, message-driven algorithms may turn mute in a system with unrestricted message loss before GST. Hence, in theory, $\diamond\Theta_{loss} \prec \diamond PS$ and $\diamond?\Theta_{loss} \prec \diamond?PS$. Since no decent distributed computing problem has a solution $\diamond\Theta_{loss}$ and $\diamond?\Theta_{loss}$, however, this result is not particularly meaningful.

Self-Stabilization

Another interesting aspect for comparison is the ability of a system to converge towards "good states" when starting from an arbitrary state. Relation 4 below reveals that all Θ-Models are weaker than their *PS* counterparts in this regard. Note that this weakness manifests itself in an impossibility result "a la FLP", i.e., a disadvantage in terms of the problems that can be solved in the Θ-Models.

More specifically, as we have shown in [28], Lemma 13 and hence Relation 3 do not hold when the requirement of self-stabilization is added to message-driven algorithms: It turned out that implementing the eventually strong failure detector $\diamond\mathcal{S}$ [11] deterministically in a self-stabilizing manner is impossible in message-driven systems with timing uncertainty, bounded memory and unbounded/unknown link capacity (i.e., the maximum number of messages that can be simultaneously in transit over a link), although time-driven implementations of $\diamond\mathcal{S}$ are known [9].

Informally, this is due to the fact that arbitrary messages could have been sent before stabilization time GST, i.e., during the erroneous period, that, however, arrive only after GST. As the only notion of time processes are able to obtain in message-driven algorithms is derived from message arrivals, those "old" messages could be indistinguishable from the correct messages that trigger the, say, ℓ-th computing step ϕ_p^ℓ at some process p after GST. Since these "old" messages could arrive earlier than the correct ones, ϕ_p^ℓ could be erroneously triggered earlier, thereby generating new erroneous messages. For certain message patterns, this can go on forever.

Relation 4 (Stabilization) *With respect to self-stabilization from an arbitrary initial state, we have*

- $\Theta \prec PS_r \approx PS$
- $?\Theta \prec ?PS_r \approx ?PS$
- $\diamond\Theta \prec \diamond PS_r \approx \diamond PS$
- $\diamond?\Theta \prec \diamond?PS_r \approx \diamond?PS$

Table 1 summarizes the relations between Θ-Models and *PS*-Models detailed in Relations 1–4.

We conclude this section by noting that it is easy to adapt the above results for the *PS*-Models to the Archimedean model [48]: Comparing the end-to-end delay bounds $\tau^- = c$ and $((2\alpha+1)\Phi+\Delta)c$ for the *PS*-Model with $\tau^- = c$ and $\tau^+ = u = s \cdot c$ for the Archimedean model reveals that they are comparable when choosing $s = (2\alpha+1)\Phi+\Delta$. Hence, all our results and observations related to the *PS*-Model are valid for the Archimedean model as well. The negative result with respect to Lemma 10 also holds, since Θ does not imply a bound on the ratio between τ^+ and minimal computing step time c, which is required by the Archimedean model.

	Θ	$?\Theta$	$\diamond\Theta$	$\diamond?\Theta$
Failure model	$\approx$	$\approx$	$\succ$	$\succ$
Added timing	$\prec$	$\prec$	$\prec$	$\prec$
Solvability	$\approx$	$\approx$	$\approx$	$\approx$
Self-stabilization	$\prec$	$\prec$	$\prec$	$\prec$

Table 1. Summary relation Θ-Models to corresponding *PS*-Models. An entry $\prec$ means that the Θ-Model is strictly weaker than its corresponding *PS*-Model, for example.

6.2 Other partially synchronous models

In this section, we briefly discuss how the Θ-Model compares to the other existing partially synchronous models. It will turn out that they are all incomparable.

First, all partially synchronous models except the *PS*-Models and the Archimedean model, in particular, the synchronous, timed asynchronous and semi-synchronous models, assume the availability of bounded drift local clocks. The same is true for most WTL models (except for the one of [26,27], where real-time clocks are replaced by partially synchronous processes) as well as the FAR model, although the drift bound need not be known here. All those models are hence more demanding than the Θ-Model, which does not need local clocks.

On the other hand, unlike the Θ-Model presented in this paper, all those models allow the minimal end-to-end delay τ^- to be zero and are hence considerably weaker than the Θ-Model in this regard. Consider the semi-synchronous models [8, 40, 41], for example, which also rely on Φ and Δ: Unlike in case of the *PS*-Models considered in Section 6.1, processes may deliver messages and send responses in zero time here. Hence, there is no lower bound $\tau^- > 0$ in the semi-synchronous models. The absence of this synchrony-enabling parameter is compensated by referring to bounded drift local clocks (timers, hardware clocks, watchdogs etc.), however: The existence of some (possibly unknown) Φ and Δ (and hence τ^+) makes it possible to reliably timeout messages, using the local real-time clock.

The WTL models [1, 3, 4, 7, 26, 34, 37] can be seen as a further "spatial" relaxation of the classic partially synchronous or semi-synchronous models. More specifically, rather than requiring the delay bound Δ to hold for all links in the system, it must hold only for a (small) subset of the links. Consequently, the WTL models are considerably weaker than the Θ-Models in this regard. Still, unlike the dynamic Θ-Models, the WTL models cannot deal with ever growing end-to-end delays, recall Section 6.1.

The FAR model [19, 20] is also considerably weaker than the Θ-Models in some very important aspect: Since the (unknown) finite average delay needs to hold on a

per-link basis only, without any need for a correlation, the FAR model is applicable to any distributed algorithm. By contrast, in the Θ-Models, we mainly focus on round-based algorithms, with broadcast-type communication (like the algorithms of Section 4): This communication pattern leads to end-to-end delays, which can indeed adequately be modeled by our Θ assumption if suitable real-time scheduling and queuing disciplines [25] are employed.

On the other hand, the FAR model is more demanding than the Θ-Model, since it employs a weak bounded drift clock for timing out round-trips and cannot cope with ever increasing delays (which would result in an infinite average). Moreover, the FAR Ω failure detector suffers from continuously increasing failure detection times in infinite runs, even in runs where some (unknown) fixed upper bound on round-trip delays exists.

As a consequence, the Θ-Models and any of the partially synchronous models discussed in this section have features which are more preferable over the others. They are hence indeed incomparable.

7 Conclusions

We introduced the dynamic Θ-Model, which allows end-to-end delays of messages simultaneously in transit to vary with time if only their ratio remains bounded. Like some classic partially synchronous models, it does not need bounded drift local clocks and assumes that the possibly unknown delay ratio Θ holds after some unknown global stabilization time. We provided a lock-step round simulation in this model, which allows to solve classic problems like consensus in presence of Byzantine failures. A novel timing transformation method was used for this purpose, which allows to prove correct and analyze Θ-algorithms designed for the dynamic model in the much simpler static Θ-Model. The Θ-Model was shown to be closer to pure asynchrony than the classic partially synchronous system models (*PS*-Models) in most aspects, although still equivalent in terms of solvable problems. Θ-algorithms are hence advantageous in terms of portability and coverage, since they are fully message-driven and timer-free, but require to continuously exchange (some) messages. The Θ-Model is hence particularly suitable in applications like VLSI Systems-on-Chip, where the latter does not matter.

Acknowledgments

We are grateful to Gérard Le Lann and Martin Hutle for many valuable discussions.

References

1. Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing Omega with weak reliability and synchrony assumptions. In *Proceeding of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC'03)*, pages 306–314, New York, NY, USA, 2003. ACM Press.
2. Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Consensus with byzantine failures and little system synchrony. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 147–155, 2006.
3. Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Stable leader election. In *DISC '01: Proceedings of the 15th International Conference on Distributed Computing*, pages 108–122. Springer-Verlag, 2001.
4. Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *Proceedings of the 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 328–337, St. John's, Newfoundland, Canada, 2004. ACM Press.
5. Daniel Albeseder. Evaluation of message delay correlation in distributed systems. In *Proceedings of the Third Workshop on Intelligent Solutions for Embedded Systems*, Hamburg, Germany, May 2005.
6. Yasser Ammar, Aurélien Buhrig, Marcin Marzencki, Benoît Charlot, Skandar Basrour, Karine Matou, and Marc Renaudin. Wireless sensor network node with asynchronous architecture and vibration harvesting micro power generator. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 287–292, New York, NY, USA, 2005. ACM.
7. Emmanuelle Anceaume, Antonio Fernández, Achour Mostéfaoui, Gil Neiger, and Michel Raynal. A necessary and sufficient condition for transforming limited accuracy failure detectors. *J. Comput. Syst. Sci.*, 68(1):123–133, 2004.
8. Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM (JACM)*, 41(1):122–152, 1994.
9. Joffroy Beauquier and Synnöve Kekkonen-Moneta. Fault-tolerance and self-stabilization: Impossibility results and solutions using self-stabilizing failure detectors. *International Journal of Systems Science*, 28(11):1177–1187, 1997.
10. Martin Biely and Josef Widder. Optimal message-driven implementation of omega with mute processes. In *Proceedings of the Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2006)*, volume 4280 of *LNCS*, pages 110–121, Dallas, TX, USA, November 2006. Springer Verlag.
11. Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.
12. Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, 1999.
13. Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.
14. Shlomi Dolev. *Self-Stabilization.* MIT Press, 2000.
15. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.
16. Jo C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5:107–119, 1991.

17. Virantha Ekanayake, IV Clinton Kelly, and Rajit Manohar. An ultra low-power processor for sensor networks. *SIGOPS Oper. Syst. Rev.*, 38(5):27–36, 2004.
18. Markus Ferringer, Gottfried Fuchs, Andreas Steininger, and Gerald Kempf. VLSI Implementation of a Fault-Tolerant Distributed Clock Generation. *IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT2006)*, pages 563–571, October 2006.
19. Christof Fetzer and Ulrich Schmid. Brief announcement: On the possibility of consensus in asynchronous systems with finite average response times. In *Proceedings of the 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, page 402, Boston, Massachusetts, 2004.
20. Christof Fetzer, Ulrich Schmid, and Martin Süßkraut. On the possibility of consensus in asynchronous systems with finite average response times. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS'05)*, pages 271–280, Washington, DC, USA, June 2005. IEEE Computer Society.
21. Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
22. Matthias Fuegger, Ulrich Schmid, Gottfried Fuchs, and Gerald Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In *Proceedings of the Sixth European Dependable Computing Conference (EDCC-6)*, pages 87–96. IEEE Computer Society Press, October 2006.
23. 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, Puerto Vallarta, Mexico, 1998. ACM Press.
24. Jean-François Hermant and Gérard Le Lann. Fast asynchronous uniform consensus in real-time distributed systems. *IEEE Transactions on Computers*, 51(8):931–944, August 2002.
25. Jean-François Hermant and Josef Widder. Implementing reliable distributed real-time systems with the Θ-model. In *Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS 2005)*, volume 3974 of *LNCS*, pages 334–350, Pisa, Italy, December 2005. Springer Verlag.
26. Martin Hutle, Dahlia Malkhi, Ulrich Schmid, and Lidong Zhou. Brief announcement: Chasing the weakest system model for implementing omega and consensus. In *Proceedings Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (formerly Symposium on Self-stabilizing Systems) (SSS 2006)*, LNCS, pages 576–577, Dallas, USA, Nov. 2006. Springer Verlag.
27. Martin Hutle, Dahlia Malkhi, Ulrich Schmid, and Lidong Zhou. Chasing the weakest system model for implementing omega and consensus. *IEEE Transactions on Dependable and Secure Computing*, 2008. (to appear).
28. Martin Hutle and Josef Widder. On the possibility and the impossibility of message-driven self-stabilizing failure detection. In *Proceedings of the Seventh International Symposium on Self Stabilizing Systems (SSS 2005)*, volume 3764 of *LNCS*, pages 153–170, Barcelona, Spain, October 2005. Springer Verlag. Appeared also as brief announcement in *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing (PODC'05)*.
29. Martin Hutle and Josef Widder. Self-stabilizing failure detector algorithms. In *Proc. IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN'05)*, pages 485–490, Innsbruck, Austria, February 2005. IASTED/ACTA Press.
30. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
31. 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. (Replaced by Research Report 28/2005, Institut für Technische Informatik, TU Wien, 2005.).
32. Jennifer Lundelius-Welch and Nancy A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.
33. Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996.
34. Dahlia Malkhi, Florin Oprea, and Lidong Zhou. Ω meets paxos: Leader election and stability without eventual timely links. In *Proceedings of the 19th Symposium on Distributed Computing (DISC'05)*, volume 3724 of *LNCS*, pages 199–213, Cracow, Poland, 2005. Springer Verlag.
35. Achour Mostefaoui, Eric Mourgaya, and Michel Raynal. Asynchronous implementation of failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'03)*, San Francisco, CA, June 22–25, 2003.
36. Achour Mostefaoui, David Powell, and Michel Raynal. A hybrid approach for building eventually accurate failure detectors. In *Proceedings of the 10th International Pacific Rim Dependable Computing Symposium (PRDC'04)*, pages 57–65. IEEE Computer Society, 2004.
37. Achour Mostéfaoui and Michel Raynal. Solving consensus using chandra-toueg's unreliable failure detectors: A general quorum-based approach. In P. Jayanti, editor, *Distributed Computing: 13th International Symposium (DISC'99)*, volume 1693 of *Lecture Notes in Computer Science*, pages 49–63, Bratislava, Slovak Republic, September 1999. Springer-Verlag GmbH.
38. Achour Mostefaoui, Michel Raynal, and Corentin Travers. Crash-resilient time-free eventual leadership. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems (SRDS 2004)*, pages 208–217. IEEE Computer Society, 2004.
39. S.M. Parkes and P. Armbruster. SpaceWire: a spacecraft onboard network for real-time communications. In *Proceedings 14th IEEE-NPSS Real Time Conference*, pages 6–10, June 2005.
40. S. Ponzio. The real-time cost of timing uncertainty: Consensus and failure detection. Master's thesis, Massachusetts Institute of Technology, May 1991.
41. Stephen Ponzio and Ray Strong. Semisynchrony and real time. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG'92)*, pages 120–135, Haifa, Israel, November 1992.
42. Ulrich Schmid and Christof Fetzer. Randomized asynchronous consensus with imperfect communications. In *22nd Symposium on Reliable Distributed Systems (SRDS'03)*, pages 361–370, Florence, Italy, October 6–8, 2003.

43. Ulrich Schmid and Andreas Steininger. Dezentrale Fehlertolerante Taktgenerierung in VLSI Chips. Research Report 69/2004, Technische Universität Wien, Institut für Technische Informatik, 2004. (Österr. Patentanmeldung A 1223/2004).
44. 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, Vienna, Austria, July 2-5, 2002.
45. T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.
46. Ivan E. Sutherland and Jo Ebergen. Computers without Clocks. *Scientific American*, 287(2):62–69, August 2002.
47. Paulo Veríssimo and António Casimiro. The timely computing base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930, August 2002.
48. Paul M.B. Vitányi. Time-driven algorithms for distributed control. Report CS-R8510, C.W.I., May 1985.
49. 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, Sorrento, Italy, October 2003. Springer Verlag.
50. Josef Widder. *Distributed Computing in the Presence of Bounded Asynchrony*. PhD thesis, Vienna University of Technology, Fakultät für Informatik, May 2004.
51. Josef Widder, Gérard Le Lann, and Ulrich Schmid. Failure detection with booting in partially synchronous systems. In *Proceedings of the 5th European Dependable Computing Conference (EDCC-5)*, volume 3463 of *LNCS*, pages 20–37, Budapest, Hungary, April 2005. Springer Verlag.
52. Josef Widder and Ulrich Schmid. Booting clock synchronization in partially synchronous systems with hybrid process and link failures. *Distributed Computing*, 20(2):115–140, August 2007.