Self-stabilizing Byzantine Clock Synchronization with Optimal Precision*

Pankaj Khanchandani[1] and Christoph Lenzen[2]

[1] Computer Engineering and Networks Laboratory (TIK), ETH Zurich,
KPANKAJ@ETHZ.CH
[2] Max Planck Institute for Informatics, Saarland Informatics Campus,
CLENZEN@MPI-INF.MPG.DE

Abstract. We revisit the approach to Byzantine fault-tolerant clock synchronization based on approximate agreement introduced by Lynch and Welch. Our contribution is threefold: (i) We provide a slightly refined variant of the algorithm yielding improved bounds on the skew that can be achieved and the sustainable frequency offsets. (ii) We show how to extend the technique to also synchronize clock rates. This permits less frequent communication without significant loss of precision, provided that clock rates change sufficiently slowly. (iii) We present a coupling scheme that allows to make these algorithms self-stabilizing while preserving their high precision. The scheme utilizes a low-precision, but self-stabilizing algorithm for the purpose of recovery.

1 Introduction

When designing synchronous distributed systems, the most fundamental question is how to generate and distribute the system clock. This task is mission critical, both in terms of performance and reliability. With ever-growing hardware complexity, reliable high-performance clocking becomes increasingly challenging; concurrently, the ramifications of clocking faults become harder to predict.

Against this background, it might be unsurprising that fault-tolerant distributed clock synchronization algorithms have found their way into real-world systems with high reliability demands: the Time-Triggered Protocol (TTP) [13] and FlexRay [9, 10] tolerate *Byzantine* (i.e., worst-case) faults and are utilized in cars and airplanes. Both of these systems derive from the classic fault-tolerant synchronization algorithm by Lynch and Welch [18], which is based on repeatedly performing approximate agreement [7] on the time of the next clock pulse. Another application domain with even more stringent requirements is hardware for spacecraft and satellites. Here, a reliable system clock is in demand despite frequent transient faults due to radiation. In addition, quartz oscillators are prone to damage during launch, making the use of less accurate, electronic oscillators preferable.

* Full paper available at arXiv.

Unfortunately, existing implementations are not *self-stabilizing,* i.e., do not guarantee automatic recovery from transient faults. This is essential for the space domain, but also highly desirable in the systems utilizing TTP or FlexRay. This claim is supported by the presence of various mechanisms that monitor the nodes and perform resets in case of observed faulty behavior in both protocols. Thus, it is of interest to devise synchronization algorithms that stabilize on their own, instead of relying on monitoring techniques: these need to be highly reliable as well, or their failure may bring down the system due to erroneous detection of or response to faults.

Accordingly, in this work we set out to answer the following questions:

1. Can the guarantees of [18] be further improved? In particular, how does the approach perform if local clock sources are less precise than quartz oscillators?
2. Under which circumstances is it useful to apply the technique also to frequencies, i.e., algorithmically adjust clock rates?
3. Can the solution be made self-stabilizing?

Our Contribution. We obtain promising answers to the above questions, in the sense that conceptually simple (i.e., implementation-friendly!) variations on the Lynch-Welch approach achieve excellent performance guarantees. Specifically, we obtain the following main results.

1. We present a refined analysis of a variant of the Lynch-Welch algorithm. We show that the algorithm converges to a steady-state error $E \in \mathcal{O}((\vartheta-1)T+U)$, where hardware clock rates are between 1 and ϑ, messages take between $d-U$ and d time to arrive at their destination, and $T \in \Omega(d)$ is the (nominal) time between consecutive clock pulses (i.e., the time required for a single approximate agreement step). This works even for very poor local clock sources: it suffices that $\vartheta \leq 1.1$, although the skew bound goes to infinity as ϑ approaches this critical value; for $\vartheta \leq 1.01$, the bound is fairly close to $2(\vartheta-1)T+4U$.[3]
2. We give a second algorithm that interleaves approximate agreement on clock *rates* with the phase (i.e., clock offset) correction scheme. If the clocks are sufficiently stable, i.e., the maximum rate of change ν of clock rates is sufficiently small, this enables to significantly extend T (and thus decrease the frequency of communication) without substantially affecting skews. Provided that ϑ is not too large and $\max\{(\vartheta-1)^2T, \nu T^2\} \ll U$, it is possible to achieve $\mathcal{O}(U)$ skew.
3. We introduce a generic approach that enables to couple either of these algorithms to FATAL [4, 5]. FATAL is a self-stabilizing synchronization algorithm, but in comparison suffers from poor performance. The coupling scheme permits to combine the best of both worlds, namely the self-stabilization properties of FATAL with the small skew of the Lynch-Welch synchronization scheme.

[3] For comparison, the critical value in [18] is smaller than 1.025, i.e., we can handle a factor 4 weaker bound on $\vartheta - 1$. Non-quartz oscillators used in space applications, where temperatures vary widely, may have ϑ close to this value, cf. [1].

On the technical side, the first two results require little innovation compared to prior work. However, it proved challenging to obtain clean, easy-to-implement algorithms that are amenable to a tractable analysis and achieve tight skew bounds. This is worthwhile for two reasons: (1) there is strong indication that the approach has considerable practical merit,[4] and (2) no readily usable mathematical analysis of the frequency correction scheme exists in the literature.[5]

In contrast, the coupling scheme we use to combine our non-stabilizing algorithms with FATAL showcases a novel technique of independent interest. We leverage FATAL's clock "beats" to effectively (re-)initialize the synchronization algorithm we couple it to. Here, care has to be taken to avoid such resets from occurring during regular operation of the Lynch-Welch scheme, as this could result in large skews or spurious clock pulses. The solution is a feedback mechanism enabling the synchronization algorithm to actively trigger the next beat of FATAL at the appropriate time. FATAL stabilizes regardless of how these feedback signals behave, while actively triggering beats ensures that all nodes pass the checks which, if failed, trigger the respective node being reset.

Organization of the paper. After presenting related work and the model, we proceed in the order of the main results listed above: phase synchronization (Section 4), frequency synchronization (Section 5), and finally the coupling scheme adding self-stabilization (Section 6). Due to space constraints, all proofs are omitted from this extended abstract in favor of conceptual descriptions.

2 Related Work

TTP [13] and FlexRay [9, 10] are both implemented in software (barring minor hardware components). This is sufficient for their application domains: the goal here is to enable synchronous communication between hardware components at frequencies in the megahertz range. Solutions fully implemented in hardware are of interest for two reasons. First, having to implement the full software abstraction dramatically increases the number of potential reasons for a node to fail – at least from the point of view of the synchronization algorithm. A slim hardware implementation is thus likely to result in a substantially higher degree of reliability of the clocking mechanism. Second, if higher precision of synchronization is required, the significantly smaller delays incurred by dedicated hardware make it possible to meet these demands.

Apart from these issues, the complexity of a software solution renders TTP and FlexRay unsuitable as fault-tolerant clocking schemes for VLSI circuits. The DARTS project [3, 11] aimed at developing such a scheme, with the goal of coming up with a robust clocking method for space applications. Instead of being based on the Lynch-Welch approach, it implements the fault-tolerant synchronization

[4] A prototype FPGA implementation achieves 182 ps skew [12], which is suitable for generating a system clock.

[5] The framework in [15, 16] addresses frequency correction, but substantial specialization would be required to achieve good constants in the bounds.

algorithm by Srikanth and Toueg [17]. Unfortunately, DARTS falls short of its design goals in two ways. On the one hand, the Srikanth-Toueg primitive achieves skews of $\Theta(d)$, which tend to be significantly larger than those attainable with the Lynch-Welch approach.[6] Accordingly, the operational frequency DARTS can sustain (without large communication buffers and communication delays of multiple logical rounds) is in the range of 100 MHz, i.e., about an order of magnitude smaller than typical system speeds. Moreover, DARTS is not self-stabilizing. This means that DARTS – just like TTP and FlexRay – is unlikely to successfully cope with high rates of transient faults. Worse, the rate of transient faults will scale with the number of nodes (and thus sustainable faults). For space environments, this implies that adding fault-tolerance without self-stabilization cannot be expected to increase the reliability of the system at all.

These concerns inspired follow-up work seeking to overcome these downsides of DARTS. From an abstract point of view, FATAL [4, 5] can be interpreted as another incarnation of the Srikanth-Toueg approach. However, FATAL combines tolerance to Byzantine faults with self-stabilization in $\mathcal{O}(n)$ time with probability $1 - 2^{-\Omega(n)}$; after recovery is complete, the algorithm maintains correct operation deterministically. Like DARTS, FATAL and the substantial line of prior work on Byzantine self-stabilizing synchronization algorithms (e.g., [2, 8]) cannot achieve better clock skews than $\Theta(d)$. The key motivation for the present paper is to combine the better precision achieved by the Lynch-Welch approach with the self-stabilization properties of FATAL.

Concerning frequency correction, little related work exists. A notable exception is the extension of the interval-based synchronization framework to rate synchronization [15, 16]. In principle, it seems feasible to derive similar results by specialization and minor adaptions of this powerful machinery to our setting. Unfortunately, apart from the technical hurdles involved, this is very likely to result in worse constants and more involved algorithms, and it is unclear whether our approach to self-stabilization can be fitted to this framework. However, it is worth noting that the overall proof strategies for the (non-stabilizing) phase and frequency correction algorithms bear notable similarities to this generic framework: separately deriving bounds on the precision of measurements, plugging these into a generic convergence argument, and separating the analysis of frequency and phase corrections.

Coming to lower bounds and impossibility results, the following is known.

- In a system of n nodes, no algorithm can tolerate $\lceil n/3 \rceil$ Byzantine faults. All mentioned algorithms are optimal in that they tolerate $\lceil n/3 \rceil - 1$ Byzantine faults [6].
- To tolerate this number of faults, $\Omega(n^2)$ communication links are required.[7] All mentioned algorithms assume full connectivity and communicate by broadcasts (faulty nodes may not adhere to this). Less well-connected topologies are outside the scope of this work.

[6] d tends to be at least one or two orders of magnitude larger than U.

[7] If a node has fewer than $2f + 1$ neighbors in a system tolerating f faults, it cannot distinguish whether it synchronizes to a group of f correct or f faulty neighbors.

– The worst-case precision of an algorithm cannot be better than $(1-1/n)U$ in a network where communication delays may vary by U [14]. In the fault-free case and with $\vartheta - 1$ sufficiently small, this bound can be almost matched (cf. Section 4); all variants of the Lynch-Welch approach match this bound asymptotically granted sufficiently accurate local clocks.
– Trivially, the worst case precision of any algorithm is at least $(\vartheta - 1)T$ if nodes exchange messages every T time units. In the fault-free case, this is essentially matched by our phase correction algorithm as well.
– With faults, the upper bound on the skew of the algorithm increases by factor $1/(1-\alpha)$, where $\alpha \approx 1/2$ if $\vartheta \approx 1$. It appears plausible that this is optimal under the constraint that the algorithm's resilience to Byzantine faults is optimal, due to a lower bound on the convergence rate of approximate agreement [7].

Overall, the resilience of the presented solution to faults is optimal, its precision asymptotically optimal, and it seems reasonable to assume that there is little room for improvement in this regard. In contrast, no non-trivial lower bounds on the stabilization time of self-stabilizing fault-tolerant synchronization algorithms are known. It remains an open question whether it is possible to achieve stabilization within $o(n)$ time.

3 Model

We assume a fully connected system of n nodes, up to $f := \lfloor (n-1)/3 \rfloor$ of which may be Byzantine faulty (i.e., arbitrarily deviate from the protocol). We denote by V the set of all nodes and by $C \subseteq V$ the subset of *correct* nodes, i.e., those that are not faulty.

Communication is by broadcast of "pulses," which are messages without content: the only information conveyed is when a node transmitted a pulse. Nodes can distinguish between senders; this is used to distinguish the case of multiple pulses being sent by a single (faulty) node from multiple nodes sending one pulse each. Note that faulty nodes are not bound by the broadcast restriction, i.e., may send a pulse to a subset of the nodes only. The system is semi-synchronous. A pulse sent by node $v \in C$ at (Newtonian) time $p_v \in \mathbb{R}_0^+$ is received by node $w \in C$ at time $t_{vw} \in [p_v + d - U, p_v + d]$; we refer to d as the *maximum message delay* (or, chiefly, delay) and to U as the *delay uncertainty* (or, chiefly, uncertainty).

For these timing guarantees to be useful to an algorithm, the nodes must have a means to measure the progress of time. Each node $v \in C$ is equipped with a hardware clock H_v, which is modeled as a strictly increasing function $H_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$. We require that there is a constant $\vartheta > 1$ such that for all times $t < t'$, it holds that $t' - t \leq H_v(t') - H_v(t) \leq \vartheta(t' - t)$, i.e., the hardware clocks have bounded drift. We remark that our results can be easily translated to the

case of discrete and bounded clocks.[8] We refer to $H_v(t)$ as the *local time* of v at time t.

Executions are event-based, where an event at node v is the reception of a message, a previously computed (and stored) local time being reached, or the initialization of the algorithm. A node may then perform computations and possibly send a pulse. For simplicity, we assume that these operations take zero time; adapting our results to account for computation time is straightforward.

Problem. A clock synchronization algorithm generates distinguished events or *clock pulses* at times $p_v(r)$ for $r \in \mathbb{N}$ and $v \in C$ so that the following conditions are satisfied for all $r \in \mathbb{N}$.

1. $\forall v, w \in C : \ |p_v(r) - p_w(r)| \leq e(r)$
2. $\forall v \in C : \ A_{\min} \leq p_v(r+1) - p_v(r) \leq A_{\max}$

The first requirement is a bound on the synchronization error between the r^{th} clock ticks; naturally, it is desired that $e(r)$ is as small as possible. The second requirement is a bound on the time between consecutive clock ticks, which can be translated to a bound on the frequency of the clocks; here, the goal is that $A_{\min}/A_{\max} \approx 1$. The *precision* of the algorithm is measured by the steady state error[9] $E := \lim_{r' \to \infty} \sup_{r \geq r'} \{e(r)\}$. Self-stabilization will be introduced and discussed in Section 6.

4 Phase Synchronization Algorithm

Our basic algorithm is a variant of the one by Lynch and Welch [18], which synchronizes clocks by simulating perpetual synchronous approximate agreement [7] on the times when clock pulses should be generated. We diverge only in terms of communication: instead of round numbers, nodes broadcast content-free pulses. Due to sufficient waiting times between pulses, during regular operation received messages from correct nodes can be correctly attributed to the respective round. In fact, the primary purpose of transmitting round numbers in the Lynch-Welch algorithm is to add recovery properties. Our technique for adding self-stabilization (presented in Section 6) leverages the pulse synchronization algorithm from [4, 5] instead, which requires to broadcast constant-sized messages only.

Properties of Approximate Agreement Steps. Abstractly speaking, the synchronization performs approximate agreement steps in each (simulated synchronous) round. In approximate agreement, each node is given an input value and the goal is to let nodes determine values that are close to each other and within the interval spanned by the correct nodes' inputs.

In the clock synchronization setting, there is the additional obstacle that the communicated values are points in time. Due to delay uncertainty and drifting

[8] Discretization can be handled by re-interpreting the discretization error as part of the delay uncertainty. All our algorithms use the hardware clock exclusively to measure bounded time differences.

[9] Typically, $e(r)$ is a monotone sequence, implying that simply $E = \lim_{r \to \infty} e(r)$.

Algorithm 1: Approximate agreement step at node $v \in \mathbf{C}$ (with synchronous message exchange).

1 // node v is given input value x_v;
2 broadcast x_v to all nodes (including self);
3 // if $w \in C$, the received value $\hat{x}_{wv} \in [x_w - \delta, x_w + \delta]$;
4 receive first value $\hat{x}_{wv}$ from each node w ($\hat{x}_{wv} := x_v$ if no message from w received);
5 $S_v \leftarrow \{\hat{x}_{wv} \,|\, w \in V\}$;
6 denote by S_v^k the k^{th} element of S_v w.r.t. ascending order;
7 $y_v \leftarrow \dfrac{S_v^{f+1} + S_v^{n-f}}{2}$;
8 **return** y_v;

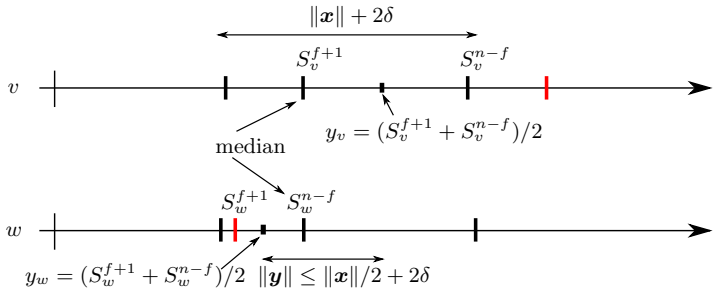

Fig. 1. An execution of Algorithm 1 at nodes v and w, where $n = 4$. There is a single faulty node, whose values are indicated in red. Note that the ranges spanned by the values received from non-faulty nodes are identical up to a perturbation of $\pm\delta$.

clocks, the communicated values are subject to a (worst-case) perturbation of at most some $\delta \in \mathbb{R}_0^+$. We will determine δ later in our analysis of the clock synchronization algorithms; we assume it to be given for now. The effect of these disturbances is straightforward: they may shift outputs by at most δ in each direction, increasing the range of the outputs by an additive 2δ in each step (in the worst case).

Algorithm 1 describes an approximate agreement step from the point of view of node $v \in C$. When implementing this later on, we need to make use of timing constraints to ensure that (i) correct nodes receive each other's messages in time to perform the associated computations and (ii) correct nodes' messages can be correctly attributed to the round to which they belong. Figure 1 depicts how a round unfolds assuming that these timing constraints are satisfied.

Denote by $\boldsymbol{x}$ the $|C|$-dimensional vector of correct nodes' inputs, i.e., $(\boldsymbol{x})_v = x_v$ for $v \in C$. The *diameter* $\|\boldsymbol{x}\|$ of $\boldsymbol{x}$ is the difference between the maximum and minimum components of $\boldsymbol{x}$. Formally, $\|\boldsymbol{x}\| := \max_{v \in C}\{x_v\} - \min_{v \in C}\{x_v\}$. We will use the same notation for other values, e.g. $\boldsymbol{y}$ and $\|\boldsymbol{y}\|$. For simplicity, we assume that $|C| = n - f$ in the following; all statements can be adapted by replacing $n - f$ with $|C|$ where appropriate.

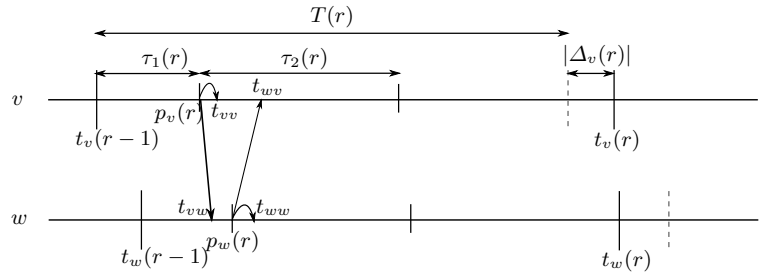

Fig. 2. A round of Algorithm 2 from the point of view of nodes v and w. Note that the durations marked on the horizontal axis are measured using the local hardware clock.

Consider the special case of $\delta = 0$. Intuitively, Algorithm 1 discards the smallest and largest f values each to ensure that values from faulty nodes cannot cause outputs to lie outside the range spanned by the correct nodes' values. Afterwards, y_v is determined as the midpoint of the interval spanned by the remaining values. Since $f < n/3$, i.e., $n - f \geq 2f + 1$, the median of correct nodes' values is part of all intervals computed by correct nodes. From this, it is easy to see that $\|\boldsymbol{y}\| \leq \|\boldsymbol{x}\|/2$, cf. Figure 1. For $\delta > 0$, we simply observe that the resulting values y_v, $v \in C$, are shifted by at most δ compared to the case where $\delta = 0$, resulting in $\|\boldsymbol{y}\| \leq \|\boldsymbol{x}\|/2 + 2\delta$.

Lemma 1. $\forall v \in C : \min_{w \in C}\{x_w\} - \delta \leq y_v \leq \max_{w \in C}\{x_w\} + \delta$.

Corollary 1. $\max_{v \in C}\{|y_v - x_v|\} \leq \|\boldsymbol{x}\| + \delta$.

Lemma 2. $\|\boldsymbol{y}\| \leq \|\boldsymbol{x}\|/2 + 2\delta$.

Algorithm. Algorithm 2 shows the pseudocode of the phase synchronization algorithm at node $v \in C$. It implements iterative approximate agreement steps on the times when to send pulses. The algorithm assumes that the nodes are initialized within a (local) time window of size F. In each round $r \in \mathbb{N}$, the nodes estimate the phase offset of their pulses[10] and then compute an according phase correction $\Delta_v(r)$. Figure 2 illustrates how a round of the algorithm plays out.

To fully specify the algorithm, we need to determine how long the waiting periods in each round are (in terms of local time), which will be given as $\tau_1(r)$, $\tau_2(r)$, and $T(r) - \Delta(r) - \tau_1(r) - \tau_2(r)$. Here, we must ensure for all $r \in \mathbb{N}$ that

1. for all $v, w \in C$, the message that v broadcasts at time $t_v(r-1) + \tau_1(r)$ is received by w at a local time from $[H_w(t_w(r-1)), H_w(t_w(r-1)) + \tau_1(r) + \tau_2(r)]$,
2. for all $v \in C$, $T(r) - \Delta_v(r) \geq \tau_1(r) + \tau_2(r)$, i.e., v computes $H_v(t_v(r))$ *before* time $t_v(r)$.

If these conditions are satisfied at all correct nodes, we say that *round r is executed correctly*, and we can interpret the round as an approximate agreement

[10] Dividing the measured local time differences by $(\vartheta + 1)/2$ is an artifact of our "one-sided" definition of hardware clock rates from $[1, \vartheta]$; in an implementation, one simply reads the hardware clocks (which exhibit symmetric error) without any scaling.

Algorithm 2: Phase synchronization algorithm at $v \in C$. Round $r+1$, $r \in \mathbb{N}$, starts at time $t_v(r)$.

```
1  // H_w(0) ∈ [0, F) for all w ∈ V
2  wait until time t_v(0) with H_v(t_v(0)) = F;
3  foreach round r ∈ ℕ do
4      start listening for messages;
5      wait until local time H_v(t_v(r−1)) + τ_1(r);        // all nodes are in round r
6      broadcast clock pulse to all nodes (including self);
7      wait until local time H_v(t_v(r−1)) + τ_1(r) + τ_2(r);// correct nodes' messages
       arrived
8      for each node w ∈ V do
9          τ_wv := H_v(t_wv), where first message from w received at t_wv (τ_wv := ∞ if
           none received);
10     S_v ← {2(τ_wv − τ_vv)/(ϑ + 1) | w ∈ V} (as multiset);
11     let S_v^k denote the k^th smallest element of S_v;
12     Δ_v(r) ← (S_v^{f+1} + S_v^{n−f}) / 2;
13     // T(r) denotes the nominal length of round r
14     wait until time t_v(r) with H_v(t_v(r)) = H_v(t_v(r−1)) + T(r) − Δ_v(r);
```

step. We will show in the next section that the following condition is sufficient for all rounds to be executed correctly.

Condition 1 *Define* $e(1) := F + (1 - 1/\vartheta)\tau_1(1)$ *and inductively for all* $r \in \mathbb{N}$ *that*

$$e(r+1) := \frac{2\vartheta^2 + 5\vartheta - 5}{2(\vartheta+1)}\, e(r) + (3\vartheta - 1)U + \left(1 - \frac{1}{\vartheta}\right)\left(T(r) + \tau_1(r+1) - \tau_1(r)\right).$$

For $r \in \mathbb{N}$, *we require* $\tau_1(r) \geq \vartheta e(r)$, $\tau_2(r) \geq \vartheta(e(r) + d)$, *and* $T(r) \geq \tau_1(r) + \tau_2(r) + \vartheta(e(r) + U)$.

Here, $e(r)$ is a bound on the synchronization error in round r, i.e., we will show that $\|\boldsymbol{p}(r)\| \leq e(r)$ for all $r \in \mathbb{N}$, provided Condition 1 is satisfied.

Analysis. In this section, we prove that Condition 1 is indeed sufficient to ensure that $\|\boldsymbol{p}(r)\| \leq e(r)$ for all $r \in \mathbb{N}$. In the following, denote by $\boldsymbol{p}(r)$, $r \in \mathbb{N}_0$, the vector of times when nodes $v \in C$ broadcast their r^{th} pulse, i.e., $H_v(p_v(r)) = H_v(t_v(r-1)) + \tau_1(r)$. If $v \in C$ takes note of the pulse from $w \in C$ in round r, the corresponding value $\tau_{wv} - \tau_{vv}$ can be interpreted as inexact measurement of $p_w(r) - p_v(r)$. This is captured by the following lemma, which provides precise bounds on the incurred error.

Lemma 3. *Suppose* $v \in C$ *receives the pulses from both* $w \in C$ *and itself in round* r *at a time from* $[H_v(t_v(r-1)), H_v(t_v(r-1)) + \tau_1(r) + \tau_2(r)]$. *Then*

$$\left| \frac{2(\tau_{wv} - \tau_{vv})}{\vartheta + 1} - (p_w(r) - p_v(r)) \right| < \vartheta U + \frac{\vartheta - 1}{\vartheta + 1} \|\boldsymbol{p}(r)\|.$$

Using Lemma 3, we can interpret the phase shifts $\Delta_v(r)$ as outcomes of an approximate agreement step with $\delta = \vartheta U + (\vartheta - 1)\|\boldsymbol{p}(r)\|/(\vartheta + 1)$, yielding the following corollary.

Corollary 2. *Suppose in round $r \in \mathbb{N}$, it holds for all $v, w \in C$ that v receives the pulse from $w \in C$ and itself in round r during $[H_v(t_v(r-1)), H_v(t_v(r-1)) + \tau_1(r) + \tau_2(r)]$. Then*

1. *$|\Delta_v(r)| < \vartheta(\|\boldsymbol{p}(r)\| + U)$ and*
2. *$\max_{v,w \in C}\{p_v(r) - \Delta_v(r) - (p_w(r) - \Delta_w(r))\} \leq (5\vartheta - 3)\|\boldsymbol{p}(r)\|/(2(\vartheta+1)) + 2\vartheta U$.*

This readily yields a bound on $\|\boldsymbol{p}(r+1)\|$ – provided that all nodes can compute when to send the next pulse on time.

Corollary 3. *Assume that round $r \in \mathbb{N}$ is executed correctly. Then*

$$\|\boldsymbol{p}(r+1)\| \leq \frac{2\vartheta^2 + 5\vartheta - 5}{2(\vartheta + 1)}\|\boldsymbol{p}(r)\| + (3\vartheta - 1)U + \left(1 - \frac{1}{\vartheta}\right) T(r)\,.$$

This bound hinges on the assumption that the round is executed correctly. Calculations show that the bounds imposed by Condition 1 are sufficient.

Lemma 4. *Suppose that $\tau_1(r) \geq \vartheta(\|\boldsymbol{p}(r)\| - (d - U))$, $\tau_2(r) \geq \vartheta(\|\boldsymbol{p}(r)\| + d)$, and that $T(r) \geq \tau_1(r) + \tau_2(r) + \vartheta(\|\boldsymbol{p}(r)\| + U)$. Then round r is executed correctly.*

We have almost all pieces in place to inductively bound $\|\boldsymbol{p}(r)\|$ and determine suitable values for $\tau_1(r)$, $\tau_2(r)$, and $T(r)$. The last missing bit is an induction anchor, i.e., a bound on $\|\boldsymbol{p}(1)\|$. This follows from the assumption that all hardware clocks are initialized within F time units of each other.

Corollary 4. $\|\boldsymbol{p}(1)\| \leq F + (1 - 1/\vartheta)\tau_1(1) = e(1)$.

Theorem 1. *If Condition 1 is satisfied, for all $r \in \mathbb{N}$, it holds that $\|\boldsymbol{p}(r)\| \leq e(r)$. If $\alpha = (6\vartheta^2 + 5\vartheta - 9)/(2(\vartheta + 1)(2 - \vartheta)) < 1$ (which holds for $\vartheta \leq 1.1$), we can choose parameters so that this is true and Algorithm 2 has steady state error $E = \lim_{r \to \infty} e(r) \leq ((\vartheta - 1)d + (4\vartheta - 2)U)/((2 - \vartheta)\alpha)$.*

5 Phase and Frequency Synchronization Algorithm

In this section, we briefly summarize our results on extending the phase synchronization algorithm to also synchronize frequencies. The basic idea is to apply the approximate agreement not only to phase offsets, but also to frequency offsets. To this end, in each round the phase difference is measured twice, applying any phase correction only after the second measurement. This enables nodes to obtain an estimate of the relative clock speeds, which in turn is used to obtain an estimate of the differences in clock speeds.

Ensuring that this procedure is executed correctly is straightforward by limiting $|\mu_v(r) - 1|$ to be small, where $\mu_v(r)$ is the factor by which node v changes its clock rate during round r. However, constraining this multiplier means that approximate agreement steps cannot be performed correctly in case $\mu_v(r+1)$ would lie outside the valid range of multipliers. This is fixed by introducing a correction that "pulls" frequencies back to the default rate.

5.1 Additional Assumptions on the Clocks

We require that clock rates satisfy a Lipschitz condition as well. In the following, we assume that H_v is differentiable (for all $v \in C$) with derivative h_v, where h_v satisfies for $t, t \in \mathbb{R}_0^+$ that $|h_v(t') - h_v(t)| \leq \nu|t' - t|$ for some $\nu > 0$. Note that we maintain the model assumption that hardware clock rates are close to 1 at all times, i.e., $1 \leq h_v(t) \leq \vartheta$ for all $t \in \mathbb{R}_0^+$.

5.2 Algorithm

Algorithm 3 gives the pseudocode of our approach. Mostly, the algorithm can be seen as a variant of Algorithm 2 that allows for speeding up clocks by factors $\mu_v(r) \in [1, \vartheta^2]$, where $\vartheta h_v(t)$ is considered the nominal rate at time t.[11] For simplicity, we fix all local waiting times independently of the round length.

The main difference to Algorithm 2 is that a second pulse signal is sent before the phase correction is applied, enabling to determine the rate multipliers for the next round by an approximate agreement step as well. A frequency measurement is obtained by comparing the (observed) relative rate of the clock of node w during a local time interval of length $\tau_2 + \tau_3$ to the desired relative clock rate of 1. Since the clock of node v is considered to run at speed $\mu_v(r)h_v(t)$ during the measurement period, the former takes the form $\mu_v(r)\Delta_{wv}/(\tau_2 + \tau_3)$, where Δ_{wv} is the time difference between the arrival times of the two pulses from w measured with H_v. The approximate agreement step results in a new multiplier $\hat{\mu}_v(r+1)$ at node v; we then move this result by ε in direction of the nominal rate multiplier ϑ and ensure that we remain within the acceptable multiplier range $[1, \vartheta^2]$.

To fully specify the algorithm, we need to determine how long the waiting periods are (in terms of local time) and choose ε. These can be determined if we ensure that *round r was executed correctly* for all $r \in \mathbb{N}$, i.e.,

1. for all $v, w \in C$, the message v broadcasts at time $t_v(r-1) + \tau_1/\mu_v(r-1)$ is received by w at a local time from $[H_w(t_w(r-1)), H_w(t_w(r-1)) + \tau_1/\mu_v(r-1) + \tau_2/\mu_w(r)]$,
2. for all $v, w \in C$, the message v broadcasts at time $t_v(r-1) + \tau_1/\mu_v(r-1) + (\tau_2+\tau_3)/\mu_v(r)$ is received by w at a local time from $[H_w(t_w(r-1)) + \tau_1/\mu_v(r-1) + \tau_2/\mu_w(r), H_w(t_w(r-1)) + \tau_1/\mu_v(r-1) + (\tau_2+\tau_3+\tau_4)/\mu_w(r)]$, and
3. for all $v \in C$, $T - \Delta_v(r) \geq \tau_1/\mu_v(r-1) + (\tau_2+\tau_3+\tau_4)/\mu_v(r)$, i.e., v computes $H_v(t_v(r))$ *before* time $t_v(r)$.

Main result. Due to space constraints, we omit the analysis, noting that the overall strategy is very similar to the one for Algorithm 2. In fact, we simply reuse the analysis from Section 4 to show that the modified algorithm executes

[11] Given that hardware clock speeds may differ by at most factor ϑ, nodes need to be able to increase or decrease their rates by factor ϑ: a single deviating node may be considered faulty by the algorithm, so each node must be able to bridge this speed difference on its own.

Algorithm 3: Phase and frequency synchronization algorithm, code for node $v \in C$. Time $t_v(r)$, $r \in \mathbb{N}_0$, is the time when round $r+1$ starts.

// $H_w(0) \in [0, F)$ for all $w \in V$
wait until time $t_v(0)$ with $H_v(t_v(0)) = F$;
// initialize clock rate multiplier
$\mu_v(0) := \mu_v(1) := \vartheta$;
foreach *round* $r \in \mathbb{N}$ **do**
  // phase correction step
  start listening for messages;
  wait until local time $H_v(t_v(r-1)) + \tau_1/\mu_v(r-1)$;
  broadcast clock pulse to all nodes (including self);
  wait until local time $H_v(t_v(r-1)) + (\tau_1 + \tau_2)/\mu_v(r)$;
  for *each node* $w \in V$ **do**
    $\tau_{wv} := H_v(t_{wv})$ (first message from w while listening at time t_{wv};
    $\tau_{wv} := \infty$ if none);
  $S_v \leftarrow \{2(\tau_{wv} - \tau_{vv})/(\vartheta + 1) \mid w \in V\}$ (as multiset);
  let S_v^k denote the k^{th} smallest element of S_v;
  $\Delta_v(r) \leftarrow \dfrac{S_v^{f+1} + S_v^{n-f}}{2}$;
  // frequency correction step
  start listening for messages;
  wait until local time $H_v(t_v(r-1)) + (\tau_1 + \tau_2 + \tau_3)/\mu_v(r)$;
  broadcast clock pulse to all nodes (including self);
  wait until local time $H_v(t_v(r-1)) + (\tau_1 + \tau_2 + \tau_3 + \tau_4)/\mu_v(r)$;
  for *each node* $w \in V$ **do**
    $\tau'_{wv} := H_v(t'_{wv})$ (first message from w while listening at time t'_{wv};
    $\tau_{wv} := \infty$ if none);
    $\Delta_{wv} := H_v(t'_{wv}) - H_v(t_{wv})$;
  $S_v \leftarrow \{1 - \mu_v(r)\Delta_{wv}/(\tau_2 + \tau_3) \mid w \in V\}$ (as multiset);
  let S_v^k denote the k^{th} smallest element of S_v;
  $\xi_v(r) \leftarrow \dfrac{S_v^{f+1} + S_v^{n-f}}{2}$;
  $\hat{\mu}_v(r+1) \leftarrow \mu_v(r) + 2\xi_v(r)/(\vartheta + 1)$;
  // pull back towards nominal frequency by ε, ensure minimum and maximum rate
  if $\hat{\mu}_v(r+1) \leq \vartheta$ **then**
    $\mu_v(r+1) \leftarrow \max\{\hat{\mu}_v(r+1) + \varepsilon, 1\}$;
  else
    $\mu_v(r+1) \leftarrow \min\{\hat{\mu}_v(r+1) - \varepsilon, \vartheta^2\}$;
  wait until time $t_v(r)$ with $H_v(t_v(r)) + (T - \Delta_v(r))/\mu_v(r)$; // nominal round length is T

rounds correctly. Then we analyze the convergence of frequencies and derive an improved skew bound by adapting Corollary 3 to yield a better bound if frequency deviations are small. This leads to the main result, which specializes to the following more readable corollary.

Corollary 5. *Suppose that* $\vartheta \leq 1.01$ *and* $\alpha := (4\vartheta^6 + 5\vartheta^3 - 7)/(2(\vartheta^3 + 1)) \approx 1/2$. *Then, for any nominal round length* T *satisfying* $T \gg F + d$ *and* $\max\{(\vartheta - 1)^2 T, \nu T^2\} \in \mathcal{O}(U)$, *a steady state error of* $\mathcal{O}(U)$ *can be achieved.*

Corollary 5 basically states that increasing T is fine, as long as $\max\{(\vartheta - 1)^2 T, \nu T^2\} \in \mathcal{O}(U)$. This improves over Algorithm 2, where it is required that $(\vartheta - 1)T \ll U$, as it permits to transmit pulses at significantly smaller frequencies (granted that ν is sufficiently small).

6 Self-stabilization

In this section, we propose a generic mechanism that can be used to transform our algorithms into *self-stabilizing* solutions; for simplicity, we assume that this algorithm is Algorithm 2 throughout this section. An algorithm is self-stabilizing, if it (re)establishes correct operation from arbitrary states in bounded time. If there is an upper bound on the time this takes in the worst case, we refer to it as the stabilization time. We stress that, while self-stabilizing solutions to the problem are known, all of them have skew $\Omega(d)$; augmenting the Lynch-Welch approach with self-stabilization capabilities thus enables to achieve an optimal skew bound of $\mathcal{O}((\vartheta - 1)T + U)$ in Byzantine self-stabilizing manner for the first time.

Our approach can be summarized as follows. Nodes locally count their pulses modulo some $M \in \mathbb{N}$. We use a low-frequency, imprecise, but self-stabilizing synchronization algorithm (called FATAL) from earlier work [4, 5] to generate a "heartbeat." On each such beat, nodes will locally check whether the next pulse with number 1 modulo M will occur within an expected time (local) window whose size is determined by the precision the algorithm would exhibit after M correctly executed pulses (in the non-stabilizing case). If this is not the case, the node is "reset" such that pulse 1 will occur within this time window.

This strategy ensures that a beat forces all nodes to generate a pulse with number 1 modulo M within a bounded time window. Assuming a value of F corresponding to its length in Algorithm 2 hence ensures that the algorithm will run as intended—at least up to the point when the next beat occurs. Inconveniently, if the beat is not synchronized with the next occurrence of a pulse 1 mod M, some or all nodes may be reset, breaking the guarantees established by the perpetual application of approximate agreement steps. This issue is resolved by leveraging a feedback mechanism provided by FATAL: FATAL offers a (configurable) time window during which a NEXT signal externally provided to each node may trigger the next beat. If this signal arrives at each correct node at roughly the same time, we can be sure that the corresponding beat is generated shortly thereafter. This allows for sufficient control on when the next beat occurs to prevent any

node from ever being reset after the first (correct) beat. Since FATAL stabilizes regardless of how the externally provided signals behave, this suffices to achieve stabilization of the resulting compound algorithm.

FATAL. We sum up the properties of FATAL in the following corollary, where each node has the ability to trigger a local NEXT signal perceived by the local instance of FATAL at any time.

Corollary 6 (of [5]). *For suitable parameters $P, B_1, B_2, B_3, D \in \mathbb{R}^+$, FATAL stabilizes within $\mathcal{O}((B_1 + B_2 + B_3)n)$ time with probability $1 - 2^{-\Omega(n)}$. Once stabilized, nodes $v \in C$ generate beats $b_v(k)$, $k \in \mathbb{N}$, such that the following properties hold for all $k \in \mathbb{N}$.*

1. *For all $v, w \in C$, we have that $|b_v(k) - b_w(k)| \leq P$.*
2. *If no $v \in C$ triggers its NEXT signal during $[\min_{w \in C}\{b_w(k)\} + B_1, t]$ for some $t \leq \min_{w \in C}\{b_w(k)\} + B_1 + B_2 + B_3$, then $\min_{w \in C}\{b_w(k+1)\} \geq t$.*
3. *If all $v \in C$ trigger their NEXT signals during $[\min_{w \in C}\{b_w(k)\} + B_1 + B_2, t]$ for some $t \leq \min_{w \in C}\{b_w(k)\} + B_1 + B_2 + B_3$, then $\max_{w \in C}\{b_w(k+1)\} \leq t + P$.*

Algorithm. Our self-stabilizing solution utilizes both FATAL and the clock synchronization algorithm with very limited interaction (rendering the approach fairly generic). We already stressed that FATAL will stabilize regardless of the NEXT signals and note that it is not influenced by Algorithm 4 in any other way. Concerning the clock synchronization algorithm, we assume that a "careful" implementation is used that does not maintain state variables for a long time. Concretely, in Algorithm 2 this is achieved by clearing memory between loop iterations.

Algorithm 4 gives the interface code, which is basically an ongoing consistency check based on the beats that resets the clock synchronization algorithm if necessary. The feedback triggering the next beat in a timely fashion is implemented by simply triggering the NEXT signal on each M^{th} beat, with a small delay ensuring that all nodes arrive in the same round and have their counter variable i reading 0. The consistency checks then ask for $i = 0$ and the next pulse being triggered within a certain local time window; if either does not apply, the reset function is called, ensuring that both conditions are met. Figure 3 visualizes how FATAL and the clock synchronization algorithm interact. Naturally, the stabilization mechanism requires R^-, R^+, and M (the parameters of Algorithm 4) to satisfy certain constraints; we refer to the full version of the paper for the respective list.

Analysis. Our analysis starts with the first correct beat produced by FATAL, which is perceived at node $v \in C$ at time $b_v(1)$. We first establish that the first beat guarantees to "initialize" the synchronization algorithm such that it will run correctly from this point on. We use this do define the "first" pulse times $p_v(1)$, $v \in C$, as well.

Lemma 5. *Let $b := \min_{v \in C}\{b_v(1)\}$. We have that*

1. *Each $v \in C$ generates a pulse at time $p_v(1) \in [b + R^-/\vartheta, b + P + R^+ + \tau_1]$.*

Algorithm 4: Interface algorithm, actions for node $v \in C$ in response to a local event at time t. Runs in parallel to local instances of FATAL and Algorithm 2.

```
// algorithm maintains local variable i ∈ {0, ..., M − 1}
if v generates a pulse at time t then
    i := i + 1 mod M;
    if i = 0 then
        wait for local time H_v(t) + ϑe(M);
        trigger NEXT signal;
if v generates a beat at time t then
    if i ≠ 0 then
        // beats should align with every M^th pulse, hence reset
        reset(R^+);
    else if next pulse would be sent before local time H_v(t) + R^− then
        // reset to avoid early pulse
        reset(R^+ − (H_v(t') − H_v(t))), where t' is the current time;
    else if next round has not started yet at local time H_v(t) + R^+ then
        // reset to avoid late pulse and start listening for other nodes' pulses on time
        reset(0);
Function reset(τ)
    halt local instance of clock synchronization algorithm;
    wait for τ local time;
    i := 0;
    H_v(t_v(0)) := H_v(t'), where t' is current time (i.e., t_v(0) := t');
    restart loop of clock synchronization algorithm (in round r = 1);
```

2. $\|\boldsymbol{p}(1)\| \leq e(1)$.
3. *At time* $p_v(1)$, $v \in C$ *sets* $i := 1$.
4. $w \in C$ *receives the pulse sent by* $v \in C$ *at a local time from* $[H_w(p_w(1)) - \tau_1, H_w(p_w(1)) + \tau_2]$.
5. *This is the only pulse* w *receives from* v *at a local time from* $[H_w(p_w(1)) - \tau_1, H_w(p_w(1)) + \tau_2]$.
6. *Denoting by round* 1 *the execution of the for-loop in Algorithm 2 during which each* $v \in C$ *sends the pulse at time* $p_v(1)$*, this round is executed correctly.*

Up to the point in time where future beats interfere, we can conclude that the synchronization algorithm will be executed as intended.

Corollary 7. *Denote by* N *the infimum over all times* $t \geq b + B_1$ *at which some* $v \in C$ *triggers a NEXT signal. If* $\min_{v \in C}\{p_v(M) + e(M)\} \leq \min\{N, b + B_1 + B_2 + B_3\}$*, then all rounds* $r \in \{1, \ldots, M\}$ *are executed correctly and* $\|\boldsymbol{p}(r)\| \leq e(r)$.

This enables to show that the first time when node $v \in C$ triggers its NEXT signal after time $b + B_1$ falls within the window of opportunity for triggering the next beat provided by FATAL.

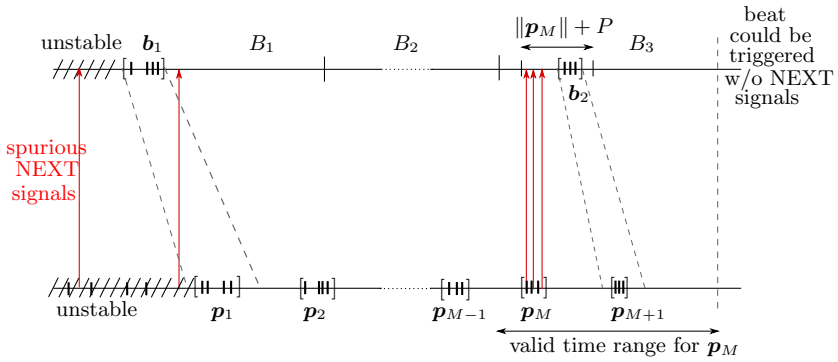

Fig. 3. Interaction of the beat generation and clock synchronization algorithms in the stabilization process, controlled by Algorithm 4. Beat $\boldsymbol{b}_1$ forces pulse $\boldsymbol{p}_1$ to be roughly synchronized. The approximate agreement steps then result in tightly synchronized pulses. By the time the nodes trigger beat $\boldsymbol{b}_2$ by providing NEXT signals based on $\boldsymbol{p}_M$, synchronization is tight enough to guarantee that the beat results in no resets.

Lemma 6. *For $v \in C$, denote by $N_v(1)$ the infimum of times $t \geq b + B_1$ when it triggers its NEXT signal. We have $H_v(N_v(1)) = p_v(M) + \vartheta e(M)$ and $b + B_1 + B_2 \leq N_v(1) \leq b + B_1 + B_2 + B_3$.*

This readily implies that the second beat occurs in response to the NEXT signals, which itself are aligned with pulse M. This yields that no correct node is reset by the second beat.

Corollary 8. *For all $v \in C$, $b_v(2) \in [p_v(M), p_v(M) + (\vartheta + 1)e(M) + P]$.*

Lemma 7. *Node $v \in C$ does not call the reset function of Algorithm 4 in response to beat $b_v(2)$.*

Repeating the above reasoning for all pairs of beats $\boldsymbol{b}(k)$, $\boldsymbol{b}(k+1)$, $k \in \mathbb{N}$, it follows that no correct node is reset by any beat other than the first. Thus, the clock synchronization algorithm is indeed (re-)initialized by the first beat to run without any further meddling from Algorithm 4.

Theorem 2. *Suppose that Algorithm 1 is executed with Algorithm 2 as synchronization algorithm. If $\vartheta \leq 1.03$, then all parameters can be chosen such that the compound algorithm self-stabilizes in $\mathcal{O}(n)$ time and has steady state error $E \leq ((\vartheta - 1)T + (3\vartheta - 1)U)/(1 - \beta)$, where $\beta = (2\vartheta^2 + 5\vartheta - 5)/(2(\vartheta + 1))$. Here, any nominal round length $T \geq T_0 \in \mathcal{O}(d_F + d)$ is possible.*

Observe that, in comparison to Theorem 1, the expression obtained for the steady state error replaces d by $\mathcal{O}(d_F + d)$, which is essentially the skew upon initialization by the first beat. Concerning stabilization, we remark that it takes $\mathcal{O}(n)$ time with probability $1 - 2^{-\Omega(n)}$, which is directly inherited from FATAL. The subsequent convergence to small skews is not affected by n, and will be much faster for realistic parameters, so we refrain from a more detailed statement.

References

1. Overview of Silicon Oscillators by Linear Technology (retrieved May 2016), http://cds.linear.com/docs/en/product-selector-card/2PB_osccalcfb.pdf
2. Daliot, A., Dolev, D.: Self-Stabilizing Byzantine Pulse Synchronization. Computing Research Repository abs/cs/0608092 (2006)
3. Distributed Algorithms for Robust Tick-Synchronization (2005–2008), http://ti.tuwien.ac.at/ecs/research/projects/darts. Research project [retrieved: 05, 2014]
4. Dolev, D., Függer, M., Lenzen, C., Posch, M., Schmid, U., Steininger, A.: Rigorously Modeling Self-Stabilizing Fault-Tolerant Circuits: An Ultra-Robust Clocking Scheme for Systems-on-Chip. Journal of Computer and System Sciences 80(4), 860–900 (2014)
5. Dolev, D., Függer, M., Lenzen, C., Schmid, U.: **F**ault-tolerant **A**lgorithms for **T**ick-generation in **A**synchronous **L**ogic: Robust Pulse Generation. Journal of the ACM 61(5), 30:1–30:74 (2014)
6. Dolev, D., Halpern, J.Y., Strong, H.R.: On the possibility and impossibility of achieving clock synchronization. Journal of Computer and System Sciences 32(2), 230–250 (1986)
7. Dolev, D., Lynch, N.A., Pinter, S.S., Stark, E.W., Weihl, W.E.: Reaching Approximate Agreement in the Presence of Faults. Journal of the ACM 33, 499–516 (1986)
8. Dolev, S., Welch, J.L.: Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults. Journal of the ACM 51(5), 780–799 (2004)
9. FlexRay Consortium, et al.: FlexRay communications system-protocol specification. Version 2.1 (2005)
10. Függer, M., Armengaud, E., Steininger, A.: Safely Stimulating the Clock Synchronization Algorithm in Time-Triggered Systems - a Combined Formal & Experimental Approach. IEEE Trans. Industrial Informatics 5(2), 132–146 (2009)
11. Függer, M., Schmid, U.: Reconciling Fault-Tolerant Distributed Computing and Systems-on-Chip. Distributed Computing 24(6), 323–355 (2012)
12. Huemer, F., Kinali, A., Lenzen, C.: Fault-tolerant Clock Synchronization with High Precision. In: IEEE Symposium on VLSI (ISVLSI) (2016), to appear
13. Kopetz, H., Bauer, G.: The Time-Triggered Architecture. Proceedings of the IEEE 91(1), 112–126 (2003)
14. Lundelius, J., Lynch, N.: An Upper and Lower Bound for Clock Synchronization. Information and Control 62(2–3), 190–204 (1984)
15. Schossmaier, K.: Interval-based Clock State and Rate Synchronization. Ph.D. thesis, Technical University of Vienna (1998)
16. Schossmaier, K., Weiss, B.: An Algorithm for Fault-Tolerant Clock State and Rate Synchronization. In: 18th Symposium on Reliable Distributed Systems (SRDS). pp. 36–47 (1999)
17. Srikanth, T.K., Toueg, S.: Optimal Clock Synchronization. Journal of the ACM 34(3), 626–645 (1987)
18. Welch, J.L., Lynch, N.A.: A New Fault-Tolerant Algorithm for Clock Synchronization. Information and Computation 77(1), 1–36 (1988)