Clock Distribution with Gradient TRIX*

Christoph Lenzen
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
lenzen@cispa.de

Shreyas Srinivas
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
shreyas.srinivas@cispa.de

Abstract

Gradient clock synchronization (GCS) algorithms minimize the worst-case clock offset between the nodes in a distributed network of diameter D and size n. They achieve optimal offsets of $\Theta(\log D)$ locally, i.e., between adjacent nodes [14] and $\Theta(D)$ globally [2]. A key open problem in this area is to achieve fault tolerance at minimal edge replication overhead.

In this work, we achieve this goal under the assumption of an average-case distribution of faults, i.e., nodes fail with independent probability $p \in o(n^{-1/2})$. In more detail, we present a self-stabilizing GCS algorithm for a grid-like directed graph with in- and out-degrees of 3. Note that even for tolerating a single fault, this degree is necessary, and if p was larger, it would not hold with probability $1 - o(1)$ that each node has at most one faulty in-neighbor. Our algorithm achieves asymptotically optimal local skew of $\Theta(\log D)$ with probability $1-o(1)$; this holds under general worst-case assumptions on link delay and clock speed variations, provided they change slowly relative to the speed of the system.

On the one hand, our results are of practical interest. As we discuss, the fault model is suitable for synchronously clocked hardware. Since our algorithm can simultaneously sustain a constant number of arbitrary changes due to faults in each clock cycle, it achieves sufficient robustness to dramatically increase the size of synchronously clocked systems.

On the other hand, our results are of a theoretical interest. We show that for a worst-case distribution of f faulty nodes within our fault model's locality constraints, our algorithm achieves local skew $O(5^f \log D)$. With probabilistically distributed faults, this becomes $O(\log D)$. Moreover, our work opens up avenues for further investigation of fault-tolerant synchronization, in particular trade-offs between fault distribution and edge density.

CCS Concepts

• **Fault Tolerance**; • **Clock Synchronisation**; • **Distributed algorithms**; • **Hardware → Very large scale integration design**;

Keywords

Clock Synchronisation, Fault Tolerance, VLSI, Self-Stabilisation

*A full version of this paper can be found on arxiv [15]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PODC '25, Huatulco, Mexico
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1885-4/25/06
https://doi.org/10.1145/3732772.3733524

ACM Reference Format:
Christoph Lenzen and Shreyas Srinivas. 2025. Clock Distribution with Gradient TRIX. In *ACM Symposium on Principles of Distributed Computing (PODC '25), June 16–20, 2025, Huatulco, Mexico.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3732772.3733524

1 Introduction

In their seminal work from 2004 [7], Fan and Lynch introduced the task of Gradient Clock Synchronization (GCS). In a distributed network with imperfect reference clocks at each process and communication channels with uncertain message delays, it requires each process to construct logical clocks with the minimum possible clock *skew*, i.e. absolute worst-case difference between clock outputs. The crucial distinction between GCS and "classic" clock synchronization frameworks is to not only consider the *global skew*, i.e., the maximum skew between any pair of nodes, but also the *local skew*, the maximum skew between neighbors.

The main insights motivating minimization of *local skew* are:

- In many cases, the skew between adjacent nodes is the appropriate measure of quality.
- The global skew grows at least linearly with the diameter D of the network [2].

The authors of [14] achieve optimal local skew with their GCS algorithm, and the authors of [3] make the algorithm resilient to localized Byzantine faults through heavy node and edge replication. However, practical applications demand fault tolerance with minimal network connectivity, which remains an open question.

Fault-Tolerant Clock Synchronization Problem (Informal)

Compute at each node of a distributed system a logical clock with the following properties.

- **Minimizing Global Skew**: The skew between any pair of nodes, i.e., the *global skew* is minimized as a function of the network diameter D: $\Theta(D)$.
- **Minimizing Local Skew**: The skew between adjacent pairs of nodes, i.e., the *local skew* is minimized as a function of the network diameter D: $\Theta(\log D)$.
- **Fault Tolerance**: An unknown set of permanently faulty nodes, upto f per neighbourhood can be tolerated, i.e., each node has at most f faulty neighbors. We call this f-local fault tolerance.
- **Self-Stabilization**: After system-wide transient (i.e., temporary) faults, the processes re-converge to optimal skews.
- **Optimal Edge Density**: Achieve the above in a network topology with minimal node degree.

This work provides a positive answer for this question for grid-like graphs and a reasonable distribution of faults, motivated by the application of clocking VLSI systems. It synthesizes two distinct lines of work, namely *gradient clock synchronization* [14] and *fault tolerant clock distribution* [5, 16] which respectively achieve optimal clock skews and low-overhead fault tolerance. Doing so requires reconciling seemingly mutually exclusive approaches of the respective works, which constitutes one of the main technical challenges of this paper. Additionally, our work is an exercise in theory building. We make modelling choices that realistically account for the kind of networks and faults that occur in VLSI circuits, accepting some limitations on the generality of the fault model for strong performance guarantees. We carefully discuss these choices, their consequences, and related open questions.

Replication for Fault Tolerance. Bund et al. [3] obtain a fault-tolerant variant of the GCS algorithm from [14] by replicating nodes and edges. They do so by simulating the (non-fault-tolerant) algorithm on the original network, replacing each node by a clique of size $3f+1$ and each edge by a biclique. The clique then synchronizes internally using the classic fault-tolerant Lynch-Welch algorithm [17], and, with some acrobatics, the resulting local outputs can be interpreted as a joint cluster clock executing the algorithm from [14] in lieu of the corresponding node of the original network. Thus, given an arbitrary network as input, one achieves gradient clock synchronization in the corresponding replicated network with up to f faults per clique.

This approach ticks many of the above boxes. Skews are asymptotically optimal, fault-tolerance is as desired, and using the self-stabilization properties of the GCS algorithm and standard techniques, it is highly plausible that self-stabilization could be achieved. However, the edge replication factor of $\Theta(f^2)$ is more than required, as a node connectivity of $2f+1$ is sufficient to maintain (possibly sub-optimal) synchronization. From a practical viewpoint, solving even the case of $f=1$ would be big step forward, but even here, the constants of the solution are poor: for $f=1$, node degrees will be at least 15.

Practical applications are typically highly sensitive to the network resources required. For example, floorplanning for VLSI systems is concerned with minimising wire length and congestion [8, Section 1.10.1] [1]. Concretely, this is a crucial concern when distributing a system clock in hardware, where the factor 5 gap to the minimum node degree of $2f+1=3$ is prohibitive: this "moderate constant" causes a headache to the engineer trying to route all of these edges with few layers and precise timing, substantially increasing communication delay uncertainty. This, in turn, directly translates into increased skews, placing the break-even point with prior art beyond relevant limits.

Clock Distribution. These considerations motivate the quest for getting as close to the minimum required connectivity as possible. This line of investigation led to the study of *fault-tolerant clock distribution* in low-degree networks [5, 16]. Both of these works have the following in common: they assume that the clock signal is generated at a central location and forwarded along by nodes in a grid-like graph. Their simple pulse forwarding schemes are self-stabilizing by design and resilient to isolated faults. The basic idea is to propagate the signal from layer to layer, having each node wait for two nodes signaling a clock pulse before locally generating and forwarding their own pulse. Moreover, it is assumed that in absence of faults, delays are changing only slowly over time. Thus, matching the input frequency to the expected delay between grid layers results in clock pulses that are well-synchronized between adjacent layers.

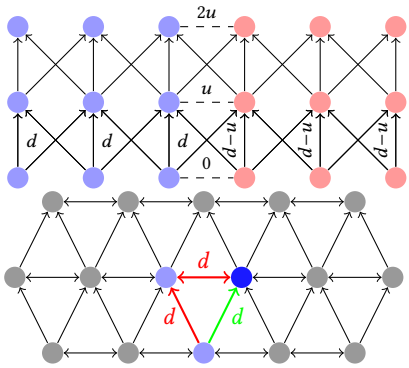


Figure 1: TRIX [16] (left) and HEX [5] (right) grids. TRIX uses the naive pulse forwarding scheme of waiting for the second copy of each pulse before forwarding it. We see how the TRIX grid can accumulate a local skew of $\Theta(uD)$ in layer D. In the HEX grid, each node waits for two copies of a pulse from in-neighbors. However, 2 of the 4 in-neighbors are on the same layer, causing a skew of d if a neighbor on the preceding layer crashes.

The above works differ in the used grid structure, cf. Figure 1, and achieved skew bounds:

- Denoting by $d-u$ and d the minimum and maximum end-to-end communication delay, in a grid of width D, [5] bounds the local skew by $d+O(u^2D/d)$. Since in practice $d \gg u$, this is a non-trivial bound. Unfortunately, the fact that $d \gg u$ also means that the additive term of d renders this bound far too large for applications. Even worse, for each fault this bound increases by another d.
- In [16], each fault adds at most u to the local skew. Observe that the used grid also has the minimum required connectivity, as each node has only 3 incoming and outgoing edges each. Alas, these advantages come at the expense of poor scaling of worst-case skews with the number of layers: on layer D, adjacent nodes may pulse up to uD time apart.

Our Contribution. In a nutshell, we provide a solution to the fault-tolerant clock synchronization problem under the assumption of a uniform distribution of faulty nodes, with a constant number of them changing their behavior concurrently, cf. Corollary 1.5. While we acknowledge the limitations in the fault model, nonetheless our results significantly advance the state of the art, cf. Table 1. Moreover, as we discuss in detail in Section 2, the fault model is well-justified in the application setting of clocking synchronous hardware.

Our results are achieved using the same grid as in [16], but with a different rule for forwarding pulses. Our novel algorithm is designed as a discrete and fault-tolerant counterpart to the GCS

method	global skew	local skew	resilience	self-stab.	graph topology
LW [17]	$O(1)$	$O(1)$	$< n/3$	no	complete ($D = 1$)
KL [9]	$O(1)$	$O(1)$	$< n/3$	yes	complete ($D = 1$)
HEX [5]	$O(dD)$	$d + O(u^2D/d)$	1-local	yes	grid-like, suboptimal degree
TRIX [16]	$O(uD^2)$	$O(uD)$	1-local	yes	grid-like, optimal degree
GCS [14]	$O(uD)$	$O(u \log D)$	crashes only	yes	arbitrary
Fault-tolerant GCS [3]	$O(uD)$	$O(u \log D)$	f-local	yes	$\Theta(f^2)$-augmented arbitrary graph[1]
Gradient TRIX **(this work)**	$O(uD)$	$O(u \log D)$	independent $p \in o(n^{-1/2})$	yes	grid-like, optimal degree
Gradient TRIX **(this work)**	$O(uD)$	$O(5^f u \log D)$	1-local, f = #faults	yes	grid-like, optimal degree

Table 1: Comparison to related work. With the exception of GCS, "resilience" refers to Byzantine fault-tolerance, i.e., worst-case behavior of faulty nodes. However, in our work the fault model is restricted in that only few faulty nodes change their behavior within a short amount of time. In turn, we are the first to simultaneously achieve optimal skew bounds, self-stabilization, and minimal degrees.

algorithm from [14]. Making this work requires substantial conceptual innovation and technical novelty. On the conceptual level, like [3], our algorithm simulates a discretized variant of the (non-fault-tolerant) GCS algorithm from [14].[2] The key difference is that we do not physically replicate nodes or edges, avoiding the corresponding overheads. Instead, we "split" the simulation of a node across time, associating only a short time slice with each simulating node, corresponding to the local forwarding of the pulse.

Abstractly, this can be phrased as follows. Taking an arbitrary *base graph* of minimum degree 2,[3] we create copies of the graph, referred to as layers. Each layer represents a "time step" of the GCS algorithm on the base graph. For each node, there is an edge from its copy on a given layer to the copies of itself and its neighbors on the next. The pulses forwarded along these edges serve very different functions:

- The pulse messages sent to copies of neigbhors correspond to the GCS algorithm's messages for estimating clock offsets to neighbors.
- The pulse messages sent between copies of the same node convey its local time from one of its copies to the next.

Note that this turns a permanently faulty node in the grid into a simulated node being faulty in a single time step only. This is of vital importance, because it enables us to rely on the self-stabilization properties of the GCS algorithm from [14]. These are implicitly shown in [10]; we prove them explicitly in the different setting of this work.

[2]Traditional presentations of GCS describe an algorithmic template, i.e., a set of constraints that must be obeyed by any GCS algorithm at all time instants $t \in \mathbb{R}_{\geq 0}$ [6]. Usually, a discretization of this template yields an algorithm for each node that acts at discrete time points to estimate the clocks of its neighbours and adjusts its clock speed. Our approach "splits a GCS node in time" across an entire grid column, inherently discretizing when (simulated) nodes act. However, we do not formally execute a simulation argument, as this would necessitate to first generalize existing (continuous) GCS schemes. Nonetheless, we refer to the underlying simulation idea for the purpose of exposition.

[3]From a theoretical point of view, the cleanest choice for the base graph might be a cycle. For the considered VLSI circuit application, for physical layout a line with replicated and connected endpoints is better.

However, by itself this does not guarantee bounded skew between correct nodes, since we also need to contain the effect of such a "transient" fault on the state of the simulated algorithm. Otherwise, a fault would increase skews arbitrarily, effectively corrupting downstream nodes: at any given node, the smallest or largest time at which a pulse from neighbors on the preceding layer is received could be determined by a faulty node. We can overcome this issue if there is at most one faulty in-neighbor. The key observation to controlling the impact of a faulty node on the pulse time lies in that it can indeed affect only one of three reception times: the (i) smallest *or* (ii) largest time at which a pulse from copies of neighbors on the previous layer is received, *or* (iii) the time at which the pulse from the copy of the node itself is received. In particular, the median of these three times lies within the interval spanned by the correct in-neighbors' pulse times. By imposing a constraint to always tie the time at which a pulse is generated closely to this median, in addition to the standard "slow" and "fast" conditions of the GCS algorithm, we can limit the local impact of a fault on skews.

In summary, we seek to simultaneously simulate a time-discrete variant of the GCS algorithm from [14], while also guaranteeing that pulse forwarding times are, up to a sufficiently small deviation, identical to median reception times plus a fixed offset. Unfortunately, no existing GCS algorithm that achieves a small local skew [11–14] can be used for this purpose as-is, since their decision rules are in conflict with the above "stick to the median" requirement.

As our main technical contribution, we resolve this conflict, simultaneously adapting the resulting algorithm to the discrete setting. To do so, we determine suitably weakened discrete variants of the slow and fast conditions introduced in [12]. In essence, we allow that a simulated node whose pulse time is ahead of all of its neighbors' pulse times to delay its next pulse by the difference to the fastest neighbor; an analogous rule applies to nodes pulsing later than all of their neighbors. From the perspective of the GCS algorithm in [14], that we build on, this constitutes a potentially arbitrarily large clock "jump," which we leverage to implement the

stick-to-the-median requirement despite the arbitrary changes in timing that faulty nodes may apply to their pulse messages. To prevent uncontrolled oscillatory behavior arising from adjacent nodes "jumping" in opposite directions, we introduce an additional condition, which we refer to as the *jump condition*. Essentially, it slightly reduces how large jumps are to avoid that uncertainty in message delays and local clock speeds cause nodes to "overswing," potentially resulting in arbitrarily large skews, cf. Figure 5.

Turning so many knobs at once meant that it was not clear that such a scheme would work. Indeed, bounding the skew of this novel algorithm turned out to be highly challenging, as jumps that delay pulses rather than speeding them up invalidate the fundamental assumption that clocks progress at rate at least 1 present in all prior work [11–14]. As a result, the main technical hurdle and contribution turned out to be proving a bound on the local skew $\mathcal{L}_\ell$ between neighbors in the same layer ℓ for the fault-free case.

Theorem 1.1. *If there are no faults, then* $\mathcal{L}_\ell \leq 4\kappa(2+\log D)$ *for all* $\ell \in \mathbb{N}$.

Here D is the diameter of the base graph, and κ is a parameter of the algorithm; choosing the input clock frequency to be $1/(2d)$ results in $\kappa \in \Theta(u + (\vartheta-1)d)$ for nodes with local reference clocks running at rates between 1 and $\vartheta > 1$. All of our results require that $d \gg u + (\vartheta-1)d$, or equivalently, that the local skew remains small compared to d. Note that if this condition does not hold, we are outside the parameter range of interest: then skews become large compared to the desired duration of a clock cycle and clock frequency has to be reduced substantially.

To address faults, we bound how they can affect timing. Due to the aforementioned stick to the median rule, we can bound the local impact of a fault on timing in terms of the local skew. However, applying this argument repeatedly, skews grow exponentially in the number of faults.

Theorem 1.2. *If there are at most* f *faulty nodes and none in layer* 0*, then* $\mathcal{L}_\ell \in O(5^f \kappa \log D)$.

While tolerating a constant number of faults is certainly better than tolerating none, this is unsatisfactory, since the requirement of one faulty in-neighbor holds with probability $1 - o(1)$ for a fairly high independent probability of $p \in o(1/\sqrt{n})$. Given that the topology we are most interested in is roughly a square grid, i.e., there are roughly $\sqrt{n}$ layers, the naive approach outlined above does not result in a non-trivial bound on the local skew for any $p \in \omega(1/n)$.

To address this issue, we provide an improved analysis exploiting that our base graph has slow-growing neighborhoods as a function of distance. As the d-hop neighborhood grows linearly with d, the number of nodes in layers $\ell' \in [\ell - n^{1/12}, \ell]$ that affect the pulse time of a node in layer ℓ is in $\Theta(n^{1/6})$. Thus, if nodes fail with probability $p \in o(1/\sqrt{n})$, the probability that there are more than 2 faulty nodes within distance $n^{1/12}$ that affect a given node is $o(1/n)$. Intuitively, this buys enough time for the self-stabilization properties of the simulated algorithm to reduce its local skew again before it spirals out of control.

Theorem 1.3. *With probability* $1 - o(1)$, $\mathcal{L}_\ell \in O(\kappa \log D)$ *for all* $\ell \in \mathbb{N}$.

The final step is to extend this bound on the local skew within a layer to one that includes adjacent nodes in different layers. As we propagate pulses layer by layer, we cannot hope to match pulse times of the k-th pulse between different layers. Instead, we match the input period to the nominal time a pulse spends on each layer. This works neatly so long as there are no *changes* in message delay, clock speed, and behavior of faulty nodes between consecutive pulses.

Theorem 1.4. *If faulty nodes do not change the timing of their output pulses, then* $\mathcal{L} \in O(\kappa \log D)$ *with probability* $1 - o(1)$.

To a large extent, this strong assumption is justified in our specific context. Clock speeds of modern systems are in the gigahertz range, and the amount of change in timing that occurs within a single clock cycle is much smaller than over the lifetime of a system [18], [8, Ch. 7]. Similarly, the by far most common faults are static faults and delay faults with a static timing profile.[4] From the point of view of the receiving node, this results in an early or late pulse, respectively, without any change in behavior between pulses. Of course, timing will still change slowly, the above benign faults will occur at some point, before which the nodes worked correctly, and some faults may be more severe. Using once more that faulty nodes' impact on timing is bounded by the local skew, the bound from Theorem 1.4 extends to a constant number of arbitrary faults in each pulse alongside small changes in delays and hardware clock speeds.

Corollary 1.5. *With probability* $1-o(1)$, $\mathcal{L} \in O(\kappa \log D)$ *even when in each pulse (i) a constant number of faulty nodes change their output behavior and timing, (ii) link delays vary by up to* $n^{-1/2}u \log D$*, and (iii) hardware clock speeds vary by up to* $n^{-1/2}(\vartheta-1)\log D$.

Finally, if all else fails, we can fall back on the ability of the pulse progation algorithm to recover from arbitrary transient faults. In constrast to the *simulated* GCS algorithm, achieving self-stabilization of the pulse propagation scheme itself is straightforward due to the directionality of the propagation. We stress that in contrast to the comparative ease at which this result is obtained, it is a very powerful and crucial fault tolerance property; this ease is the result of the key design decision to propagate pulses in a directed fashion.

Theorem 1.6. *The pulse propagation algorithm can be implemented in a self-stabilizing way. It stabilizes within* $O(\sqrt{n})$ *pulses.*

In light of these results, we view this work as a major step towards simultaneously achieving high performance and strong robustness in the practical setting of clock distribution in hardware. In alignment with the theoretical question motivating this work, we achieve an asymptotically optimal local skew at the minimum possible node degree under the assumption of node failures with probability $o(n^{-1/2})$.

The Bigger Picture. The above discussion left open how our work advances the state of the art in a broader context. There are several points we would like to highlight in this context.

(1) The fault model is more limited than proven necessary by known impossibilities. In the specific application context,

[4] That is, a given timing profile of input transitions results in the same relative timing of output transitions.

this is well-motivated: there is no attacker inducing a worst-case distribution of faults or coordinating the actions of faulty nodes. However, from a theoretical point of view, a minimal set of assumptions is desirable. We view our work as a first and important step in this direction. As discussed above, reducing the skews introduced in the wake of faulty nodes required substantial changes to the algorithm and far-reaching adjustments to the proof strategy. A logical next step would be to explore whether fault-tolerant gradient clock synchronization is feasible with an undirected degree-3 network. Achieving this is likely to be necessary to overcome the limitation that only few changes in timing between consecutive pulses can be handled by our solution. We exploit the directional propagation to make self-stabilization simple and re-interpret permanent faults as transient ones from the perspective of simulating the non-fault-tolerant GCS algorithm, but this renders the above downside inherent to our scheme.

(2) We consider an average-case distribution of faults. For low-degree networks, this is a must: if an adversary can choose the position of faults, very few faults can overcome the connectivity of the network. By justifying this restriction for an important practical application, we demonstrate the utility of this modeling choice. We hope that this will inspire follow-up work on fault-tolerant clock synchronization in average-case fault models.

(3) We establish that the case of $f = 1$ can be handled with minimal node in-degrees of 3. This may open up the way towards a general scheme achieving resilience to f local faults with in-degree $2f + 1$.

(4) Our main technical contribution might prove useful in different contexts. Prior work [11–14] did not allow for adjusting clocks "backwards," which is pivotal in proving the self-stabilization properties of the simulated GCS algorithm. We anticipate that our novel techniques will prove useful in different settings. For instance, one could seek to modify the algorithm from [14] to recover from transient faults of bounded duration without increasing the local skew between non-faulty nodes by more than a constant factor.

Organization of this Article. In Section 2, we discuss the system model, introduce the graph on which we run our synchronization algorithm, and motivate our modeling choices, including its non-standard aspects. We then present a simplified version of the algorithm that better highlights the conceptual approach in Section 3. The analysis section introduces the basic elements of the analysis, while deferring most of the details to the full paper[15].

2 Modeling

We use a non-standard model, which is motivated by the specific setting outlined in the introduction. Accordingly, we will emphasize and discuss model choices where this seems prudent.

Setting. Recall that our motivating application is to provide a synchronized clock signal to a large System-on-Chip. Physically, this means that we need to provide the clock signal to a rectangular area; for simplicity, we will assume the most common case of it being square. We want to supply a uniform grid of nodes in the square area with this signal, which then will serve as roots of relatively small local clock trees supplying the low-level components with the clock signal. If these trees contribute a maximum clock skew of Δ and the skew between adjacent grid points is at most $\mathcal{L}$, the triangle inequality guarantees a worst-case skew of $\mathcal{L} + 2\Delta$ between adjacent components of the System-on-Chip. The local clock trees can be designed using standard methodology. Therefore, in the following we will focus exclusively on the grid of their roots.

A key assumption we make is that communication delay between correct adjacent nodes changes only slowly with time. This enables us to generate synchronized pulses at all grid nodes by matching the input frequency with the (inverse) propagation time between consecutive layers. This is justified for two reasons:

- The dominant sources of uncertainty in propagation delay are inaccuracies in component fabrication, aging, and temperature and frequency variations that are slow relative to the time it takes to propagate an input clock pulse across even a large System-on-Chip [18]. For example temperature-induced delay variations occur on the order of milliseconds [8, Section 7.2.2]
- Changing delays of *all* links between a pair of adjacent layers by up to δ increases skew bounds by at most δ as shown in the full version of the paper.

In order to generate sufficiently synchronized pulses at the nodes of layer 0, a straightforward solution is to use a simple path fed by the clock source, see [15, Appendix A]. This scheme suffices for our fault model,[5] since the probability that there is any (permanent) fault in layer 0 is $o(1)$ due to the small total number of nodes in a single layer. As we show in [15, Appendix A], this approach is self-stabilizing and results in well-synchronized input pulses.

In a perfect grid, all layers would consist of a path. Unfortunately, this results in the issue that the endpoints of the path, lacking one neighbor, would have only two adjacent nodes in the preceding and subsequent layer. A naive solution is to insert additional edges between the boundary nodes, turning the layer into a cycle and the entire graph into a cylinder (with some special treatment of layer 0). However, realizing such a solution on the square would result in far too long edges between boundary nodes or require to, essentially, replicate each layer, effectively doubling the number of nodes and edges in the graph.

Instead, we choose to replicate the boundary nodes only, which then provides the "missing" input to the next layer. Note that this increases the degree of the nodes next to the boundary nodes by one. We cope with this by a general analysis allowing for the layers to be copies of an arbitrary base graph of minimum degree 2. In Figures 2 and 3, we show the base graph and the connectivity of nodes between adjacent layers of our synchronization network in our assumed setting, respectively.

Network Graph. We are given a simple connected base graph $H = (V, E)$ of minimum degree 2 and diameter $D \in \mathbb{N}_{>0}$. For $v, w \in V$, denote by $d(v, w) \leq D$ the distance from v to w in H. To derive the graph $G = (V_G, E_G)$ we use for synchronization, for each

[5] Tolerating one local fault is also straightforward by using a redundant path; as this does not increase the resilience of the system asymptotically, we stick to the simplest scheme.

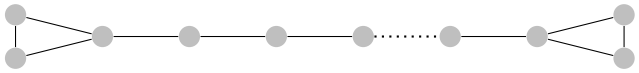


Figure 2: Base graph H used in this work. Rather than using a cycle, which would result in a TRIX grid, we replicate the end nodes of a line to ensure a minimum degree of 2. Alternatively, one could use a line and exploit that the probability that one of the $O(\sqrt{n})$ boundary nodes fails is $o(1)$.

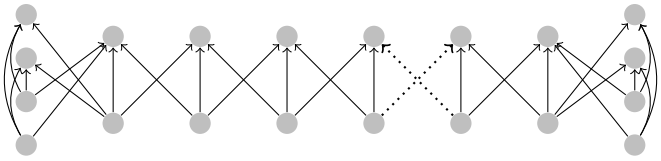


Figure 3: Layer structure of G resulting from our choice of H. Most nodes have in- and out-degree 3, some 4.

$\ell \in \mathbb{N}$ we create a copy V_ℓ of V. Denoting by (v,ℓ) the copy of $v \in V$ in V_ℓ, we define $E_\ell := \{((v,\ell),(w,\ell+1)) \mid \{v,w\} \in E \vee v = w\}$. We now obtain G by setting $V_G := \bigcup_{\ell\in\mathbb{N}} V_\ell$ and $E_G := \bigcup_{\ell\in\mathbb{N}} E_\ell$. That is, for each *layer* $\ell \in \mathbb{N}$ we have a copy of $v \in V$, which has outgoing edges to the copies of itself and all its neighbors on layer $\ell+1$. Here, ℓ is bounded from above by some value in $\Theta(\sqrt{n})$. We slightly abuse notation by neglecting this bound on ℓ in lemma statements and proofs. Since G is a DAG, we refer to out-neighbors as *successors* and in-neighbors as *predecessors*.

Fault Model. An unknown subset $F \subset V_G$ is *faulty*, meaning that these nodes do not adhere to the protocol. Edge faults are mapped to node faults, i.e., if edge $((v,\ell),(w,\ell+1))$ is faulty, we instead consider (v,ℓ) (or $(w,\ell+1)$, if preferred) faulty. We assume that each node fails independently with probability $p \in o(1/\sqrt{n})$.[6] In particular, this entails that with probability $1-o(1)$, no node has two faulty predecessors, i.e., faults are 1-local. We assume this to be the case throughout our analysis. Thus, for all $\ell \in \mathbb{N}$ and $v \in V$, $|(\{(v,\ell)\} \cup \bigcup_{\{v,w\}\in E}\{(w,\ell)\}) \cap F| \le 1$.

Faulty nodes behave arbitrarily, subject to the constraint that at most a constant number of faulty nodes change their timing behavior between consecutive pulses.

Communication. Each node has the ability to broadcast pulse messages on its outgoing edges. If node $v_\ell \in V_\ell$ broadcasts at time $t_{v,\ell}$, its successors receive its message at (potentially different) times from $[t_{v,\ell}+d-u, t_{v,\ell}+d]$. The maximum end-to-end *delay* d includes any delay caused by computation. Typically, the delay *uncertainty* u is much smaller than d. As discussed above, we assume delays to be static (or changing at a negligible rate cf. Corollary 1.5), i.e., each edge $e = ((v,\ell),(w,\ell+1))$ has an unknown, but fixed associated delay $\delta_e \in [d-u, d]$ applied to each pulse sent from (v,ℓ) to $(w,\ell+1)$.

[6] We stress that this requirement is not stronger than that of [5, 16] and [3] for $f = 1$ in any practical sense. If faults correlate in a way that they cluster together, it is likely that neighbors fail. Assuming independence (or, more generally, negative correlation) captures "faults do not cluster" in the most straightforward way that allows us to exploit this property beyond immediate neighbors.

Note that faulty nodes can send pulses at arbitrary times, without being required to broadcast; even if physical node implementations disallow point-to-point communication, edge faults could still result in this behavior.

Local Clocks and Computations. Each node is able to approximately measure the progress of time by means of a local time reference. We model this by node (v,ℓ) having query access to a *hardware clock* $H_{v,\ell}\colon \mathbb{R}_{\ge 0} \to \mathbb{R}_{\ge 0}$ satisfying

$$\forall t < t' \in \mathbb{R}_{\ge 0},\ t' - t \le H_{v,\ell}(t') - H_{v,\ell}(t) \le \vartheta(t'-t).$$

for some $\vartheta > 1$. No known phase relation is assumed between the hardware clocks. The algorithm will use them exclusively to measure how much time passes between local events. As for delays, we assume that hardware clock speeds are static (or changing slowly). This is justified in the same way as for delays.

Computations are deterministic. However, in addition to receiving a message, the hardware clock reaching a time value previously determined by the algorithm can also trigger computations and possibly the broadcast of a pulse.

Output and Skew. The goal of the algorithm is to synchronize the *pulses* generated by correct nodes. Our measure of quality is the worst-case *local skew* the algorithm guarantees. We define the local skew as the largest offset between the k-th pulses of adjacent nodes on the same layer or pulses k and $k+1$ of adjacent nodes on layers ℓ and $\ell+1$, whichever is larger. Formally, for $\ell \in \mathbb{N}$ we define

$$\mathcal{L}_\ell := \sup_{k\in\mathbb{N}} \max_{\substack{\{v,w\}\in E\\ (v,\ell),(w,\ell)\notin F}} \{|t^k_{v,\ell} - t^k_{w,\ell}|\},$$

$$\mathcal{L}_{\ell,\ell+1} := \sup_{k\in\mathbb{N}} \max_{\substack{((v,\ell),(w,\ell+1))\in E_\ell\\ (v,\ell),(w,\ell+1)\notin F}} \{|t^{k+1}_{v,\ell} - t^k_{w,\ell+1}|\},$$

and $\mathcal{L} := \sup_{\ell\in\mathbb{N}} \max\{\mathcal{L}_\ell, \mathcal{L}_{\ell,\ell+1}\}$. This deviates from the standard definition of the local skew:

- The definition is adjusted to pulse synchronization, which can be viewed as an essentially equivalent time-discrete variant of clock synchronization [6].
- Between consecutive layers, we synchronize consecutive pulses. After initialization, which is complete once the first pulse propagated through the grid, this is equivalent to a layer-dependent index shift of pulse numbers.

We assume that correct nodes on layer 0 generate well-synchronized pulses at times $t^k_{v,0}$ for $k \in \mathbb{N}_{>0}$ at a frequency of our choice. For our purposes, it suffices that $\mathcal{L}_0 \le \kappa$. We discuss how to ensure this in [15, Appendix A]. Other correct nodes generate pulses $t^k_{v,\ell}$, $k \in \mathbb{N}_{>0}$, based on the pulse messages received from their predecessors.

Any frequency error of layer 0 translates to an increase in $\mathcal{L}_{\ell,\ell+1}$ for all ℓ. For the sake of notational simplicity, we assume the frequency provided by layer 0 to match the one we choose perfectly, subsuming its error instead in the drift of the hardware clocks, i.e., ϑ. In other words, whatever drives the frequency of layer 0 is defining the "true" time t. In practice, this has little effect, since the time reference used will be based on the best clock available in the system.

Algorithm 1 Simplified pseudocode for discrete GCS at node (v, ℓ), $\ell > 0$. As shown in [15, Lemma B2], this code is equivalent to [15, Algorithm 3] in the absence of faults. The parameters Λ and κ will be determined later, based on the analysis.

```
loop
    H_own, H_min, H_max := ∞
    do
        if received pulse from (v, ℓ − 1) then
            H_own := H_{v,ℓ}(t)
        if received pulse from first (w, ℓ − 1), {v, w} ∈ E then
            H_min := H_{v,ℓ}(t)
        if received pulse from last (w, ℓ − 1), {v, w} ∈ E then
            H_max := H_{v,ℓ}(t)
    until H_own, H_min, H_max < ∞
    C_{v,ℓ} := min_{s∈ℕ}{max{H_own − H_max + 4sκ, H_own − H_min − 4sκ}} − κ/2
    if C_{v,ℓ} < 0 then
        C_{v,ℓ} := min{H_own − H_min − κ/2 + 2κ, 0}
    else if C_{v,ℓ} > ϑκ then
        C_{v,ℓ} := max{H_own − H_max − κ/2 − κ, ϑκ}
    wait until H_{v,ℓ}(t) = H_own + Λ − d − C_{v,ℓ}
    broadcast pulse
```

3 Algorithm

In this section, we discuss the pulse forwarding algorithm. We provide a simplified version of the algorithm that behaves identical so long as the predecessors of the executing node are correct. The full algorithm needs to handle the possibility that faulty nodes send multiple messages or none at all. This complicates bookkeeping and loop control, distracting from the principles underlying the algorithm's operation. Accordingly, we defer the full algorithm to appendix A of the full version[15], where we show the equivalence to the simplified variant when there are no faulty predecessors.

3.1 Simplified Pulse Forwarding Algorithm

The algorithm proceeds in iterations corresponding to pulses. In each iteration, node (v, ℓ)

(1) timestamps the arrival times of the pulses of its predecessors using its hardware clock,
(2) determines a correction value $C_{v,\ell}$ based on these timestamps, and
(3) forwards the pulse $\Lambda - d - C_{v,\ell}$ time after receiving the pulse from $v_{\ell-1}$, measured by its hardware clock.

If all reception times are close to each other, then $C_{v,\ell}$ will be small. Recalling that messages are in transit for roughly d time, this translates to Λ being the nominal time for a pulse to propagate from layer $\ell - 1$ to layer ℓ. We need to choose Λ large enough such that the above sequence can be always realized. That is, we need to consider how far apart the reception times of messages from the previous layer can be, and ensure that $\Lambda - d$ exceeds this value plus the resulting correction $C_{v,\ell}$. Assuming that this precondition holds, Algorithm 1 implements the above approach. In each loop iteration, it initializes three reception times to ∞:

- H_{own}, which stores the arrival time of the pulse from $(v, \ell-1)$. From the perspective of the simulated GCS algorithm, this reflects the state of the node $v \in V$ simulated by (v, ℓ), $\ell \in \mathbb{N}$.
- $H_{\min}$, which stores the minimum arrival time of a pulse from a neighbor $w_{\ell-1}$, $w \neq v$. This corresponds to the first pulse received from a neighbor w of v in G in this iteration.
- $H_{\max}$, which stores the maximum arrival time of a pulse from a neighbor $w_{\ell-1}$, $w \neq v$. This corresponds to the last pulse received from a neighbor w of v in G in this iteration.

The do-until loop fills these variables with the correct values. At the heart of the algorithm lies the computation of $C_{v,\ell}$. If there were no faults, one could always compute

$$\Delta := \min_{s\in\mathbb{N}} \{\max\{H_{\text{own}} - H_{\max} + 4s\kappa, H_{\text{own}} - H_{\min} - 4s\kappa\}\} - \frac{\kappa}{2}$$

and then choose the closest value from the range $[0, \vartheta\kappa]$, i.e., set (i) $C_{v,\ell} := 0$ if $\Delta < 0$, (ii) $C_{v,\ell} := \vartheta\kappa$ if $\Delta > \vartheta\kappa$, and (iii) $C_{v,\ell} := \Delta$ else.

To get intuition on this choice, observe that $\min_{x\in\mathbb{R}}\{\max\{H_{\text{own}} - H_{\max} + x, H_{\text{own}} - H_{\min} - x\}\}$ is attained when $H_{\text{own}} - H_{\max} + x = H_{\text{own}} - H_{\min} - x$. This is equivalent to $x = (H_{\max} - H_{\min})/2$, i.e., if κ was infinitesimally small, we had that $H_{\text{own}} - \Delta = (H_{\max} + H_{\min})/2$. Moreover, if each node could accurately determine the time each pulse received by it was sent, the reception times of the pulse messages could serve as exact proxies for the actual pulse forwarding times of the nodes on layer $\ell - 1$. In iteration k, this would mean to generate the pulse at (v, ℓ) faster if $(v, \ell-1)$ generated its pulse later than the average of $\min_{\{v,w\}\in E}\{t^k_{w,\ell-1}\}$ and $\max_{\{v,w\}\in E}\{t^k_{w,\ell-1}\}$. Thus, any (v, ℓ) for which $t^k_{v,\ell-1} - \min_{\{v,w\}\in E}\{t^k_{w,\ell-1}\} > \max_{\{v,w\}\in E}\{t^k_{w,\ell-1}\} - t^k_{v,\ell-1}$ would choose $C_{v,\ell} > 0$, attempting to reduce $\max_{\{v,w\}\in E}\{|t^k_{v,\ell} - t^k_{w,\ell}|\}$ compared to $\max_{\{v,w\}\in E}\{|t^k_{v,\ell-1} - t^k_{w,\ell-1}|\}$. This can be viewed as trying to reduce the local skew by a greedy strategy.

Unfortunately, this naive strategy fails to account for inaccuracies due to message delay uncertainty and drifting hardware clocks. Nonetheless, we follow this strategy up to deviations of $O(\kappa)$. The additional terms serve the following purposes:

- Considering only discrete choices for $x \in 4\kappa\mathbb{N}$ rather than arbitrary $x \in \mathbb{R}$ is the key ingredient that makes the algorithmic approach succeed, cf. [12]. Essentially, this is necessary because there is no way to determine $t_{v_{\ell-1},k} - t_{w_{\ell-1},k}$ precisely. Discretizing observed skews in units of $\kappa \in \Theta(u + (\vartheta - 1)(\Lambda - d))$ enables a delicate strategy that alternates between overestimating skews to locally generate the next pulse earlier for the sake of "catching up" with others and underestimating skews to "wait" for others catch up.
- Substracting $\kappa/2$ accounts for errors in measuring skews, which are caused by uncertainty in message delay and hardware clock speed.
- To limit the damage done by a faulty predecessor of (v, ℓ), we ensure that (v, ℓ) generates its pulse without too large of a deviation from the *median* of $t_{v,\ell-1}$, $\min_{\{v,w\}\in E}\{t^k_{w,\ell-1}\}$, and $\max_{\{v,w\}\in E}\{t^k_{w,\ell-1}\}$ (plus the nominal offset of Λ). This is achieved by permitting corrections $C_{v,\ell} < 0$ if $(v, \ell - 1)$ clearly generated its pulse earlier than $\min_{\{v,w\}\in E}\{t^k_{w,\ell-1}\}$

and $C_{v,\ell} > \vartheta\kappa$ if it clearly generated its pulse later than $\max_{\{v,w\}\in E}\{t^k_{w,\ell-1}\}$, respectively.

To further motivate the last point, recall that there can be at most one fault among the predecessors of (v,ℓ). A single faulty predecessor can affect only one of the three values H_{own}, $H_{\min}$, and $H_{\max}$: control H_{own} arbitrarily, $H_{\min}$ to be smaller than the minimum reception time from a correct node $(w,\ell-1)$, $\{v,w\}\in E$, or $H_{\max}$ to exceed the maximum reception time from correct nodes $(w,\ell-1)$, $\{v,w\}\in E$. Hence, ensuring that pulses are generated with only a small offset relative to median $\{H_{\text{own}}, H_{\min}, H_{\max}\} + \Lambda - d$ indeed limits the damage that a fault can do.

Achieving all of the desired properties is non-trivial, leading to the fairly involved choice of $C_{v,\ell}$. It can be viewed as simultaneously implementing relaxed fast and slow conditions (as introduced in [12]), an additional jump condition required to make the GCS algorithm work under these relaxed fast and slow conditions, and the requirement to stick close to the median of predecessors' pulse times. In Section 4.1, we specify the (relaxed) slow and fast condition, as well as the jump condition, and show that the algorithm implements them. [15, Lemmas 4.27 and 4.28] show that the algorithm also enforces that pulses deviate little from the time interval spanned by correct predecessors (offset by Λ).

There is some freedom in the choice of parameters. For simplicity, we fix a good choice of κ and note that d must satisfy a lower bound $B \in O(\sup_{\ell\in\mathbb{N}}\{\mathcal{L}_\ell\}+\kappa)$. Observe that this constraint simply means that the skew bounds are useful, as a skew that is of similar size as the maximum end-to-end delay requires to slow the system down substantially. Finally, Λ must be at least $d + O(\sup_{\ell\in\mathbb{N}}\{\mathcal{L}_\ell\})$, which due to the previous constraint holds e.g. for the choice $\Lambda = 2d$. Formally, for a sufficiently large constant C,

$$\kappa := 2\left(u + \left(1 - \frac{1}{\vartheta}\right)(\Lambda - d)\right), \tag{1}$$

$$\Lambda \geq C\vartheta(\sup_{\ell\in\mathbb{N}}\{\mathcal{L}_\ell\} + u) + d, \text{ and} \tag{2}$$

$$d \geq C(\vartheta(\sup_{\ell\in\mathbb{N}}\{\mathcal{L}_\ell\} + u) + \kappa). \tag{3}$$

Complete Algorithm. The complete algorithm cannot wait for messages from all predecessors to determine when to send its pulse, as a faulty node not sending its pulse then would deadlock all its descendants. As discussed above, the hardware clock time of the next pulse time does not deviate *much* from median $\{H_{\text{own}}, H_{\min}, H_{\max}\}+\Lambda-d$, but does depend on $\max\{H_{\min}, H_{\text{own}}, H_{\max}\}$ in some cases. However, we will prove that $\mathcal{L}_{\ell-1}$ is small enough such that all pulse messages from correct nodes will be received in time. Hence, it is sufficient to wait until median $\{H_{\text{own}}, H_{\min}, H_{\max}\} + \vartheta\mathcal{L}_{\ell-1}$ (or later) according to $H_{v,\ell}$. Provided that $\Lambda - d$ is large enough, this implies that any message for computing $C_{v,\ell}$ missing is due to a fault; in fact, at the point in time when this becomes clear, $C_{v,\ell}$ is already determined, regardless of how late the message would arrive.

The complete algorithm differs from Algorithm 1 by covering the case that a signal does not arrive in time. Intuitively, one can treat the respective message arrival time (H_{own} or $H_{\max}$, $H_{\min}$ is not possible) as ∞, while allowing such an ∞ to cancel out in substraction:

- If $H_{\text{own}} = \infty$, then $C_{v,\ell} \in H_{\text{own}} - H_{\max} - O(\kappa)$, and (v,ℓ) will generate its pulse at local time $H_{\text{own}} + \Lambda - d - C_{v,\ell} \in H_{\max} + \Lambda - d + O(\kappa)$.
- If $H_{\max} = \infty$ and $H_{\text{own}} \geq H_{\min}$, then $C_{v,\ell} \in H_{\text{own}} - H_{\min} \pm \Theta(\kappa)$ and (v,ℓ) will generate its pulse at local time $H_{\text{own}} + \Lambda - d - C_{v,\ell} \in H_{\min} + \Lambda - d \pm O(\kappa)$.
- If $H_{\max} = \infty$ and $H_{\text{own}} < H_{\min}$, then $C_{v,\ell} \in [0, 2\kappa]$ and (v,ℓ) will generate its pulse at local time $H_{\text{own}} + \Lambda - d - C_{v,\ell} \in H_{\text{own}} + \Lambda - d - O(\kappa)$.

Note that in all cases, the pulse is generated with an offset of $\Lambda - d - \Theta(\kappa)$ from the median reception time. The complete algorithm follows the above intuition, leveraging the fact that there is no need to wait indefinitely to determine that the missing signal is late, and is given in [15, Appendices B and C].

Last, but not least, it is of interest to make the pulse forwarding algorithm self-stabilizing [4]. Due to the design choice of propagating the clock signal from a single source along a DAG, this will immediately translate to the overall scheme being self-stabilizing, so long as the clock generation is self-stabilizing, too. This is straightforward, because one can assume that the signals from the previous layer are already well-synchronized. Thus, all that nodes need to do is to detect when all but possibly one (faulty) pulse signal arrive in close temporal proximity to determine when to clear their memory and start a new iteration of the main loop. In [15, Appendix C], we discuss how this can be achieved using standard techniques.

4 Analysis

We now outline the key elements of the analyis of our pulse progagation scheme under the assumption that layer 0 generates well-synchronized pulses: specifically the slow, fast, and jump conditions which are obeyed by Algorithm 1. We specify and provide some intuition for these conditions. A full analysis of how the algorithm obeys them and how this results in skew bounds and fault tolerance is deferred to the full paper[15] We discuss a suitable method for achieving synchronisation in layer 0 in [15, Appendix A]. Our analysis proceeds along the following lines:

(1) We show that, if the local skew is small enough compared to Λ, i.e., Equation (2) holds, all correct nodes execute their iterations as intended. That is, each correct node on layer $\ell > 0$ receives the k-th pulses of its correct predecessors in its k-th loop iteration. This is deferred to the full paper. We then proceed under the assumption that this holds true, which will be justified retroactively once we establish that the local skew is bounded.
(2) Since delays and hardware clock speeds are (approximated as being) static, any (substantial) change in relative timing of consecutive pulses is due to faulty nodes. Thus, the task of bounding the local skew reduces to bounding the intra-layer skew $\mathcal{L}_\ell$ for a single pulse, since such a bound must take into account the full variability introduced by faulty nodes. This reasoning is deferred to the full paper.
(3) Based on potentials, we analyze $\mathcal{L}_\ell$ in the absence of faults. The results entail not only bounded skew, but also that the potentials recover if they become unexpectedly large.

(4) We show that faulty nodes have limited impact on the potentials. From this and the above recovery property, we infer that skews behave favorably also when there are faults.

As stated above, the first two steps of our line of reasoning are deferred to the appendix, alongside some basic helper lemmas given in [15, Appendix D]. The main challenge is to bound $\mathcal{L}_\ell$ for a single pulse. Due to the first step, we know that the k-th pulse at correct nodes depends only on the k-th pulses of their predecessors ([15, Lemma B.1]). Therefore, in the following fix k and denote the k-th pulse time of correct $(v,\ell) \in V_G$ by $t_{v,\ell}$.

Recall that for $v, w \in V$, we denote by $d(v,w)$ their distance in the base graph H. Our analysis is built around the following potential functions.

Definition 4.1 (Potential Functions)**.** Let $v, w \in V$ and $s, \ell \in \mathbb{N}$. We define

$$\begin{aligned}\psi^s_{v,w}(\ell) &:= t_{v,\ell} - t_{w,\ell} - 4s\kappa d(v,w)\\ \Psi^s(\ell) &:= \max_{v,w\in V}\{\psi^s_{v,w}(\ell)\},\\ \xi^s_{v,w}(\ell) &:= t_{v,\ell} - t_{w,\ell} - (4s-2)\kappa d(v,w), \text{ and}\\ \Xi^s(\ell) &:= \max_{v,w\in V}\{\Xi^s_{v,w}(\ell)\}.\end{aligned}$$

Bounding $\Psi^s(\ell)$ readily translates to bounding $\mathcal{L}_\ell$.

Observation 4.2. *If for $s, \ell \in \mathbb{N}$ and some $\Psi^s \in \mathbb{R}_{\geq 0}$ it holds that $\Psi^s(\ell) \leq \Psi^s$, then $\mathcal{L}_\ell \leq \Psi^s + 4s\kappa$.*

Proof. Fix $k \in \mathbb{N}$ and suppose that $\{v,w\} \in E$ maximizes $|t_{v,\ell} - t_{w,\ell}|$. W.l.o.g., assume that $t_{v,\ell} \geq t_{w,\ell}$. Since $\{v,w\} \in E$, we have that $d(v,w) = 1$. Hence, $|t_{v,\ell} - t_{w,\ell}| = t_{v,\ell} - t_{w,\ell} = \psi^s_{v,w}(\ell) + 4s\kappa \leq \Psi^s(\ell) + 4s\kappa \leq \Psi^s + 4s\kappa$. Since $k \in \mathbb{N}$ is arbitrary, it follows that $\mathcal{L}_\ell \leq \Psi^s + 4s\kappa$. □

In summary, the goal of our analysis will be to bound $\Psi^s(\ell)$ by a small value for some s satisfying $4s\kappa \in O(u \log D)$.

We first study the behavior of the algorithm if there are no faults. Accordingly, this will be tacitly assumed in all statements of this section, with the expection of [15, Section 4.3]. Note that by [15, Lemma 4.27], this means that we may also tacitly assume that Algorithm 1 is run by all nodes in layers $\ell \in \mathbb{N}_{>0}$. In [15, Section 4.3], we will then bound the impact of faulty layers on the potential.

4.1 The Slow, Fast, and Jump Conditions

The key to bounding the local skew without faults is to find the right balance between two conflicting goals: choosing $C_{v,\ell}$ large enough to "catch up" to predecessors $w_{\ell-1} \neq v_{\ell-1}$ that generated their pulse earlier than $v_{\ell-1}$, but small enough to "wait" for predecessors $w_{\ell-1} \neq v_{\ell-1}$ that generated their pulse later than $v_{\ell-1}$. The following condition, illustrated in Figure 4, captures what we need regarding the latter.

Definition 4.3 (Slow Condition)**.** For all $s \in \mathbb{N}$, correct layers $\ell - 1 \in \mathbb{N}$, and $v_\ell \in V_\ell \setminus F$, we require the *slow condition* $\mathbf{SC}(s) :=$ **SC-1**$(s) \vee$ **SC-2**$(s) \vee$ **SC-3** to hold, where

$$\begin{aligned}\textbf{SC-1}(s)&\colon \frac{C_{v,\ell}}{\vartheta} \leq t_{v,\ell-1} - \max_{\{v,w\}\in E}\{t_{w,\ell-1}\} + 4s\kappa\\ \textbf{SC-2}(s)&\colon \frac{C_{v,\ell}}{\vartheta} \leq t_{v,\ell-1} - \min_{\{v,w\}\in E}\{t_{w,\ell-1}\} - 4s\kappa\\ \textbf{SC-3}&\colon C_{v,\ell} \leq 0.\end{aligned}$$

This can be viewed as a variant of the slow condition from [12], adjusted to our setting by quantifying by how much v_ℓ may safely shift the timing of its pulse. The main conceptual difference with [12] is that we relax the slow condition by adding **SC-3**. In what follows, we drop s from the notation when it is clear from context.

The fast condition, also illustrated in Figure 4, is the counterpart to Definition 4.3 addressing the need to "catch up" to neighbors that are ahead.

Definition 4.4 (Fast Condition)**.** For all $s \in \mathbb{N}_{>0}$, correct layers $\ell - 1 \in \mathbb{N}_{>0}$, and $v_\ell \in V_\ell \setminus F$, we require the *fast condition* $\mathbf{FC}(s) :=$ **FC-1**$(s) \vee$ **FC-2**$(s) \vee$ **FC-3** to hold, where

$$\begin{aligned}\textbf{FC-1}(s)&\colon C_{v,\ell} \geq t_{v,\ell-1} - \max_{\{v,w\}\in E}\{t_{w,\ell-1}\} + (4s-2)\kappa + \kappa\\ \textbf{FC-2}(s)&\colon C_{v,\ell} \geq t_{v,\ell-1} - \min_{\{v,w\}\in E}\{t_{w,\ell-1}\} - (4s-2)\kappa + \kappa\\ \textbf{FC-3}&\colon C_{v,\ell} \geq \kappa.\end{aligned}$$

This can be viewed as a variant of the fast condition from [12], adjusted to our setting by quantifying by how much v_ℓ may safely shift the timing of its pulse. The main conceptual difference with [12] is that we relax the fast condition by adding **FC-3**.

In addition, note that there is an additive term of κ that does not change sign. Its purpose is to account for the fact that our simulation of the GCS algorithm from [14] operates in discrete time steps corresponding to the layers. The continuous versions of the GCS algorithm in [11, 12, 14] can choose this term arbitrarily small. In contrast, we need it to exceed the maximum error in time measurement accumulated in a step. We remark that, in principle, one could choose this term different from κ. However, since both need to meet the same lower bound of $u + (1 - 1/\vartheta)(\Lambda - d)$, there is no asymptotic gain in introducing a separate parameter.

Our relaxation of the slow and fast conditions adds a substantial complication. From the perspective of the time-continuous variant of the algorithm in [12], we now allow for arbitrarily large clock "jumps," rather than bounded clock rates. In our discrete version, the rate bound from [12] corresponds to $C_{v,\ell} \in [0, \vartheta\kappa]$. Without this additional constraint, the slow and fast conditions are insufficient to bound skews.

This is illustrated in Figure 5, showing an execution that satisfies **SC** and **FC**, but suffers from skews that grow without bound. The key issue is that adjacent nodes could "jump" in opposite directions, resulting in an oscillatory behavior in which measurement errors accumulate indefinitely. To avoid this kind of behavior, we add an additional condition that "dampens" such oscillations, yet limits by how much a faulty predecessor can cause an increase in skew.

Definition 4.5 (Jump Condition)**.** For all correct layers $\ell - 1 \in \mathbb{N}_{>0}$ and $v_\ell \in V_\ell \setminus F$, we require the *jump condition*

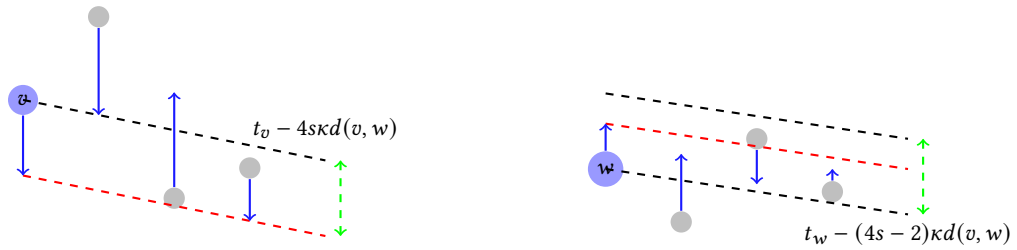


Figure 4: Slow condition (left) and fast condition (right). SC(s) is tailored to ensuring that $\max_{w\in V}\{\psi^s_{v,w}(\ell)\}$ (the length of the green arrow) cannot grow quickly. Nodes w with $C_{w,\ell} \le 0$ (SC-3 holds) cannot apply a correction pushing them below the red line. If $C_{w,\ell} > 0$, then both SC-1 and SC-2 will ensure that there is a neighbor x of w such that the offset of $t_{w,\ell-1} - C_{w,\ell}/\vartheta$ to the black line does not exceed the one of $t_{x,\ell-1}$. In other words, SC ensures that the blue arrows indicating $C_{w,\ell}/\vartheta$ do not reach below the red line. This means that any increase of $\max_{w\in V}\{\psi^s_{v,w}(\ell)\}$ is caused by delay and clock speed variation, which in turn is bounded by $\kappa/2$ per layer. Similarly, FC(s) is tailored to ensuring that $\max_{v\in V}\{\xi^s_{v,w}(\ell)\}$ (the length of the green arrow), if positive, decreases by at least $\kappa/2$. To ensure this, $C_{w,\ell}$ (indicated by blue arrows) must be large enough to reach below the red line. This is achieved by FC(s) having an additional "slack" term of κ, which overcomes the "loss" of $\kappa/2$ due to uncertainty.

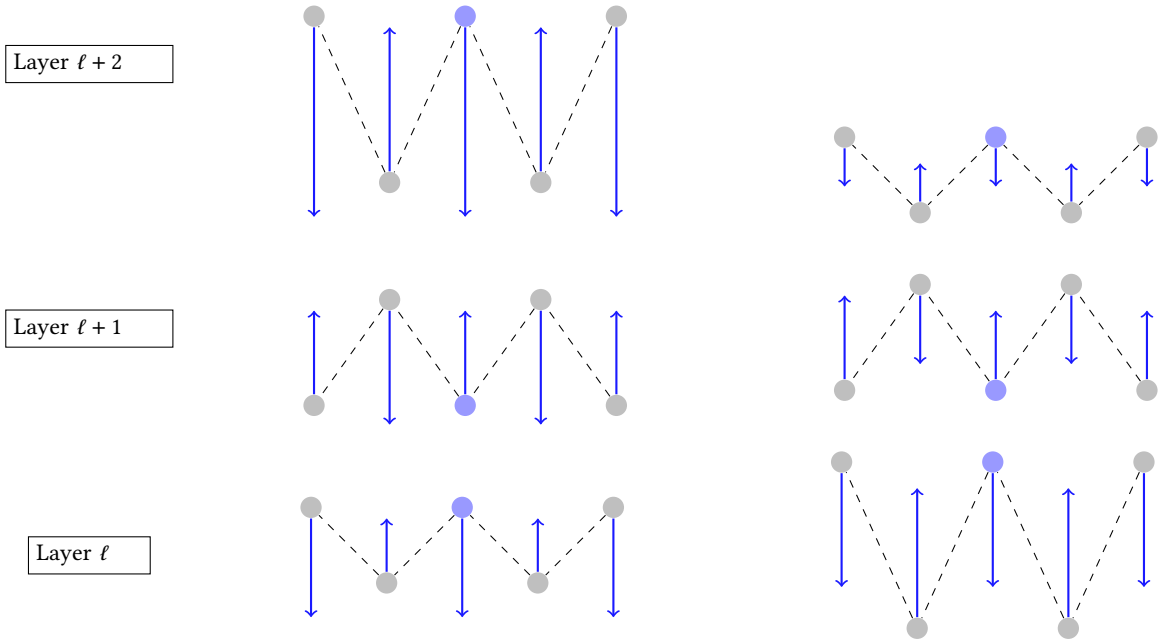


Figure 5: On the left, it is shown how skews increase without JC. While SC(0) disallows that (v,ℓ) speeds up its pulse by more than the equivalent of $(v,\ell-1)$ matching the earliest pulse of any $(w,\ell-1)$, $\{v,w\}\in E$, FC permits that a node (v,ℓ) with slow $(v,\ell-1)$ to "overshoot," i.e., $C_{v,\ell}$ (shown as blue arrow) gets large. This results in an amplifying oscillatory behavior. On the right, the same scenario is shown with JC in effect. JC forces the corrections to stop κ before the earliest or latest neighbor, respectively, resulting in a dampened oscillation.

JC := **JC-1** $\vee$ **JC-2** $\vee$ **JC-3** to hold, where

$$\textbf{JC-1}\colon \kappa < \frac{C_{v,\ell}}{\vartheta} \le t_{v,\ell-1} - \max_{\{v,w\}\in E}\{t_{w,\ell-1}\} - \kappa$$

$$\textbf{JC-2}\colon 0 > C_{v,\ell} \ge t_{v,\ell-1} - \min_{\{v,w\}\in E}\{t_{w,\ell-1}\} + \kappa$$

$$\textbf{JC-3}\colon 0 \le \frac{C_{v,\ell}}{\vartheta} \le \kappa.$$

We prove that the slow, fast, and jump condition are correctly implemented in [15, Appendix D].

Acknowledgments

This work was supported in part by the European Research Council (ERC) through the European Union's Horizon 2020 Research and Innovation Programme under Grant 716562. The author Shreyas Srinivas is also a member of the Saarbrücken Graduate School of Computer Science

References

[1] Sandeep N. Bhatt and Frank Thomson Leighton. 1984. A framework for solving VLSI graph layout problems. *J. Comput. System Sci.* 28, 2 (1984), 300–343. doi:10.1016/0022-0000(84)90071-0

[2] Saâd Biaz and Jennifer Lundelius Welch. 2001. Closed Form Bounds for Clock Synchronization under Simple Uncertainty Assumptions. *Inform. Process. Lett.* 80 (2001), 151–157.

[3] Johannes Bund, Christoph Lenzen, and Will Rosenbaum. 2019. Fault Tolerant Gradient Clock Synchronization. In *Symposium on Principles of Distributed (PODC)*. 357–365.

[4] Edsger W. Dijkstra. 1974. Self-stabilizing systems in spite of distributed control. *Commun. ACM* 17, 11 (1974), 943–644.

[5] Danny Dolev, Matthias Függer, Christoph Lenzen, Martin Perner, and Ulrich Schmid. 2016. HEX: Scaling Honeycombs is Easier than Scaling Clock Trees. *J. Comput. System Sci.* 82, 5 (2016), 929–956.

[6] Danny Dolev and Christoph Lenzen. 2021, retrieved on 04 Jan 2023. Clock Synchronisation and Adversarial Fault Tolerance. https://www.mpi-inf.mpg.de/fileadmin/inf/d1/teaching/summer21/csaft/reading-material-ch09.pdf.

[7] Rui Fan and Nancy Lynch. 2004. Gradient Clock Synchronization. In *Symposium on Principles of Distributed Computing (PODC)*. 320–327.

[8] David Harris and N Weste. 2010. CMOS VLSI Design. *ed: Pearson Education, Inc* (2010).

[9] Pankaj Khanchandani and Christoph Lenzen. 2018. Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision. *Theory of Computing Systems* (2018).

[10] Fabian Kuhn, Christoph Lenzen, Thomas Locher, and Rotem Oshman. 2010. Optimal Gradient Clock Synchronization in Dynamic Networks. *CoRR* abs/1005.2894 (2010).

[11] Fabian Kuhn, Christoph Lenzen, Thomas Locher, and Rotem Oshman. 2010. Optimal Gradient Clock Synchronization in Dynamic Networks. *Symposium on Principles of distributed computing (PODC)* (2010).

[12] Fabian Kuhn and Rotem Oshman. 2009. Gradient Clock Synchronization Using Reference Broadcasts. In *Conference on Principles of Distributed Systems (OPODIS)*. 204–218.

[13] Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. 2008. Clock Synchronization with Bounded Global and Local Skew. In *Symposium on Foundations of Computer Science (FOCS)*. 509–518.

[14] Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. 2010. Tight Bounds for Clock Synchronization. *J. ACM* 57, 2 (2010).

[15] Christoph Lenzen and Shreyas Srinivas. 2023. Clock Distribution with Gradient TRIX. arXiv:2301.05073 [cs.DC] https://arxiv.org/abs/2301.05073

[16] Christoph Lenzen and Ben Wiederhake. 2020. TRIX: Low-Skew Pulse Propagation for Fault-Tolerant Hardware. arXiv:2010.01415 https://arxiv.org/abs/2010.01415.

[17] Jennifer Lundelius Welch and Nancy A. Lynch. 1988. A New Fault-Tolerant Algorithm for Clock Synchronization. *Information and Computation* 77, 1 (1988), 1–36.

[18] Thucydides Xanthopoulos (Ed.). 2009. *Clocking in Modern VLSI Systems*. Springer US.