Self-stabilising Byzantine Clock Synchronisation is Almost as Easy as Consensus

Christoph Lenzen · clenzen@mpi-inf.mpg.de
Department of Algorithms and Complexity,
Max Planck Institute for Informatics,
Saarland Informatics Campus

Joel Rybicki · joel.rybicki@helsinki.fi
Metapopulation Research Centre,
Department of Biosciences,
University of Helsinki

Abstract. We give fault-tolerant algorithms for establishing synchrony in distributed systems in which each of the n nodes has its own clock. Our algorithms operate in a very strong fault model: we require self-stabilisation, i.e., the initial state of the system may be arbitrary, and there can be up to $f < n/3$ ongoing Byzantine faults, i.e., nodes that deviate from the protocol in an arbitrary manner. Furthermore, we assume that the local clocks of the nodes may progress at different speeds (clock drift) and communication has bounded delay. In this model, we study the pulse synchronisation problem, where the task is to guarantee that eventually all correct nodes generate well-separated local pulse events (i.e., unlabelled logical clock ticks) in a synchronised manner.

Compared to prior work, we achieve *exponential* improvements in stabilisation time and the number of communicated bits, and give the first sublinear-time algorithm for the problem:

- In the deterministic setting, the state-of-the-art solutions stabilise in time $\Theta(f)$ and have each node broadcast $\Theta(f \log f)$ bits per time unit. We exponentially reduce the number of bits broadcasted per time unit to $\Theta(\log f)$ while retaining the same stabilisation time.
- In the randomised setting, the state-of-the-art solutions stabilise in time $\Theta(f)$ and have each node broadcast $O(1)$ bits per time unit. We exponentially reduce the stabilisation time to $\operatorname{polylog} f$ while each node broadcasts $\operatorname{polylog} f$ bits per time unit.

These results are obtained by means of a recursive approach reducing the above task of *self-stabilising* pulse synchronisation in the *bounded-delay* model to *non-self-stabilising* binary consensus in the *synchronous* model. In general, our approach introduces at most logarithmic overheads in terms of stabilisation time and broadcasted bits over the underlying consensus routine.

Contents

1 Introduction

Many of the most fundamental problems in distributed computing relate to timing and fault tolerance. Even though most distributed systems are inherently asynchronous, it is often convenient to design such systems by assuming some degree of synchrony provided by reliable global or distributed clocks. For example, the vast majority of existing Very Large Scale Integrated (VLSI) circuits operate according to the synchronous paradigm: an internal clock signal is distributed throughout the chip neatly controlling alternation between computation and communication steps. Of course, establishing the synchronous abstraction is of high interest in numerous other large-scale distributed systems, as it makes the design of algorithms considerably easier.

However, as the accuracy and availability of the clock signal is typically one of the most basic assumptions, clocking errors affect system behavior in unpredictable ways that are often hard – if not impossible – to tackle at higher system layers. Therefore, *reliably* generating and distributing a joint clock is an essential task in distributed systems. Unfortunately, the cost of providing fault-tolerant synchronisation and clocking is still poorly understood.

1.1 Pulse synchronisation

In this work, we study the *self-stabilising Byzantine pulse synchronisation* problem [13, 16], which requires the system to achieve synchronisation despite severe faults. We assume a fully connected message-passing system of n nodes, where

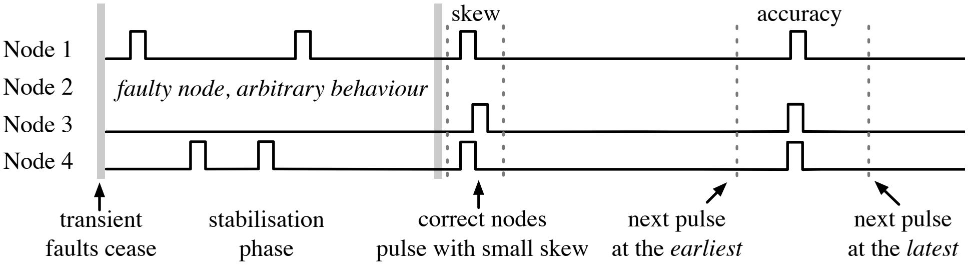
1. an unbounded number of transients faults may occur anywhere in the network, and
2. up to $f < n/3$ of the nodes can be faulty and exhibit *arbitrary* ongoing misbehaviour.

In particular, the transient faults may arbitrarily corrupt the state of the nodes and result in loss of synchrony. Moreover, the nodes that remain faulty may deviate from any given protocol, behave adversarially, and collude to disrupt the other nodes by sending them *different* misinformation even after transient faults have ceased. Note that this also covers faults of the communication network, as we may map faults of communication links to one of their respective endpoints. The goal is now to (re-)establish synchronisation once transient faults cease, despite up to $f < n/3$ Byzantine nodes. That is, we need to consider algorithms that are simultaneously (1) self-stabilising [7, 15] and (2) Byzantine fault-tolerant [24].

More specifically, the problem is as follows: after transient faults cease, no matter what is the initial state of the system, the choice of up to $f < n/3$ faulty nodes, and the behaviour of the faulty nodes, we require that after a bounded *stabilisation time* all the *non-faulty* nodes must generate pulses that

- occur almost simultaneously at each correctly operating node (i.e., have small *skew*), and
- satisfy specified minimum and maximum frequency bounds (*accuracy*).

While the system may have arbitrary behaviour during the initial stabilisation phase due to the effects of transient faults, eventually the above conditions provide synchronised unlabelled clock ticks for all non-faulty nodes:


skew
accuracy
Node 1
Node 2
faulty node, arbitrary behaviour
Node 3
Node 4
transient faults cease
stabilisation phase
correct nodes pulse with small skew
next pulse at the *earliest*
next pulse at the *latest*

In order to meet these requirements, it is necessary that nodes can estimate the progress of time. To this end, we assume that nodes are equipped with (continuous, real-valued) hardware clocks that run at speeds that may vary arbitrarily within 1 and ϑ, where $\vartheta \in O(1)$. That is, we normalize minimum clock speed to 1 and assume that the clocks have drift bounded by a constant. Observe that in an asynchronous system, i.e., one in which communication and/or computation may take unknown and unbounded time, even perfect clocks are insufficient to ensure any relative timing guarantees between the actions of different nodes. Therefore, we additionally assume that the nodes can send messages to each other that are received and processed within at most $d \in \Theta(1)$ time. The clock speeds and message delays can behave adversarially within the respective bounds given by ϑ and d.

In summary, this yields a highly adversarial model of computing, where further restrictions would render the task infeasible:

1. transient faults are arbitrary and may involve the entire network,
2. ongoing faults are arbitrary, cover erroneous behavior of both the nodes and the communication links, and the problem is not solvable if $f \geq n/3$ [11], and
3. the assumptions on the accuracy of local clocks and communication delay are minimal to guarantee solvability.

1.2 Background and related work

If one takes any one of the elements described above out of the picture, then this greatly simplifies the problem. Without permanent faults, the problem becomes trivial: it suffices to have all nodes follow a designated leader. Without transient faults [23], straightforward solutions are given by elegant classics [33, 34], where [34] also guarantees asymptotically optimal skew [29]. Taking the uncertainty of unknown message delays and drifting clocks out of the equation leads to the so-called digital clock synchronisation problem [3, 14, 25, 26, 28], where communication proceeds in synchronous rounds and the task is to agree on a consistent (bounded) round counter. While this abstraction is unrealistic as a basic system model, it yields conceptual insights into the pulse synchronisation problem in the bounded-delay model. Moreover, it is useful to assign numbers to pulses after pulse synchronisation is solved, in order to get a fully-fledged shared system-wide clock [27].

In contrast to these relaxed problem formulations, the pulse synchronisation problem was initially considered to be very challenging – if not impossible – to solve. In a seminal article, Dolev and Welch [16] proved otherwise, albeit with an algorithm having an impractical exponential stabilisation time. In a subsequent line of work, the stabilisation time was reduced to polynomial [6] and then linear in f [9]. However, the linear-time algorithm relies on simulating multiple instances of synchronous *consensus* algorithms [30] concurrently, which results in a high communication complexity.

The consensus problem [24, 30] is one of the fundamental primitives in fault-tolerant computing. Most relevant to this work is synchronous binary consensus with (up to f) Byzantine faults. Here, node v is given an input $x(v) \in \{0,1\}$, and it must output $y(v) \in \{0,1\}$ such that the following properties hold:

1. **Agreement:** There exists $y \in \{0,1\}$ such that $y(v) = y$ for all correct nodes v.
2. **Validity:** If for $x \in \{0,1\}$ it holds that $x(v) = x$ for all correct nodes v, then $y = x$.
3. **Termination:** All correct nodes eventually decide on $y(v)$ and terminate.

In this setting, two of the above main obstacles are not present: the system is properly initialised (no self-stabilisation required) and computation proceeds in synchronous rounds, i.e., well-ordered

time	bits	type	notes	reference
poly f	$O(\log f)$	det.		[6]
$O(f)$	$O(f \log f)$	det.		[9]
$O(f)$	$O(\log f)$	det.		this work and [4]
$2^{O(f)}$	$O(1)$	rand.	adversary cannot predict coin flips	[16]
$O(f)$	$O(1)$	rand.	adversary cannot predict coin flips	[13]
polylog f	polylog f	rand.	private channels, (*)	this work and [22]
$O(\log f)$	poly f	rand.	private channels	this work and [18]

Table 1: Summary of pulse synchronisation algorithms for $f \in \Theta(n)$. For each respective algorithm, the first two columns give the stabilisation time and the number of bits broadcasted by a node per time unit. The third column denotes whether algorithm is deterministic or randomised. The fourth column indicates additional details or model assumptions. All algorithms tolerate $f < n/3$ faulty nodes except for (*), where we have $f < n/(3 + \varepsilon)$ for any constant $\varepsilon > 0$.

compute-send-receive cycles. This confines the task to understanding how to deal with the interference from Byzantine nodes. Synchronous consensus is extremely well-studied; see e.g. [32] for a survey. It is known that precisely $\lfloor (n-1)/3 \rfloor$ faults can be tolerated in a system of n nodes [30], $\Omega(nf)$ messages need to be sent in total [10], the connectivity of the communication network must be at least $2f + 1$ [8], deterministic algorithms require $f + 1$ rounds [1, 20], and randomised algorithms can solve the problem in constant expected time [18]. In constrast, no non-trivial lower bounds on the time or communication complexity of pulse synchronisation are known.

The linear-time pulse synchronisation algorithm in [9] relies on simulating (up to) one synchronous consensus instance for each node simultaneously. Accordingly, this protocol requires each node to broadcast $\Theta(f \log f)$ bits per time unit. Moreover, the use of *deterministic* consensus is crucial, as failure of any consensus instance to generate correct output within a prespecified time bound may result in loss of synchrony, i.e., the algorithm would fail *after* apparent stabilisation. In [13], these obstacles were overcome by avoiding the use of consensus by reducing the pulse synchronisation problem to the easier task of generating at least one well-separated “resynchronisation point”, which is roughly uniformly distributed within any period of $\Theta(f)$ time. This can be achieved by trying to initiate such a resynchronisation point at random times, in combination with threshold voting and locally checked timing constraints to rein in the influence of Byzantine nodes. In a way, this seems much simpler than solving consensus, but the randomisation used to obtain a suitable resynchronisation point strongly reminds of the power provided by shared coins [2, 3, 18, 31] – and this is exactly what the core routine of the expected constant-round consensus algorithm from [18] provides.

1.3 Contributions

Our main result is a framework that reduces pulse synchronisation to an arbitrary (non-self-stabilising) synchronous binary consensus routine at very small overheads. In other words, given *any* efficient algorithm that solves consensus in the standard synchronous model of computing, we show how to obtain an efficient algorithm that solves the pulse synchronisation problem in the bounded-delay model with clock drift.

The key to efficiency is a recursive approach, where each node participates in only $\lceil \log f \rceil$ consensus instances, one for each level of recursion. On each level, the overhead of the reduction over a call to the consensus routine is a constant multiplicative factor both in time and bit complexity; concretely, this means that both complexities increase by overall factors of $O(\log f)$. Applying suitable consensus routines yields *exponential improvements* in bit complexity of deterministic and time complexity of randomised solutions, respectively:

1. In the deterministic setting, we exponentially reduce the number of bits each node broadcasts per time unit to $\Theta(\log f)$, while retaining $\Theta(f)$ stabilisation time. This is achieved by employing the phase king algorithm [4] in our construction.

2. In the randomised setting, we exponentially reduce the stabilisation time to polylog f, where each node broadcasts polylog f bits per time unit. This is achieved using the algorithm by King and Saia [22]. We note that this slightly reduces resilience to $f < n/(3 + \varepsilon)$ for any fixed constant $\varepsilon > 0$ and requires private communication channels.

3. In the randomised setting, we can also obtain a stabilisation time of $O(\log f)$, polynomial communication complexity, and optimal resilience of $f < n/3$ by assuming private communication channels. This is achieved using the consensus routine of Feldman and Micali [18]. This almost settles the open question by Ben-Or et al. [3] whether pulse synchronisation can be solved in expected constant time.

The running times of the randomised algorithms (2) and (3) hold with high probability and the additional assumptions on resilience and private communication channels are inherited from the employed consensus routines. Here, private communication channels mean that Byzantine nodes must make their decision on which messages to sent in round r based on knowledge of the algorithm, inputs, and all messages faulty nodes receive up to and including round r. The probability distribution is then over the independent internal randomness of the correct nodes (which the adversary can only observe indirectly) and any possible randomness of the adversary. Our framework does not impose these additional assumptions: stabilisation is guaranteed for $f < n/3$ on each recursive level of our framework as soon as the underlying consensus routine succeeds (within prespecified time bounds) constantly many times in a row. Our results and prior work are summarised in Table 1.

Regardless of the employed consensus routine, we achieve a skew of $2d$, where d is the maximum message delay. This is optimal in our model, but overly pessimistic if the sum of communication and computation delay is not between 0 and d, but from (d^-, d^+), where $d^+ - d^- \ll d^+$. In terms of d^+ and d^-, a skew of $\Theta(d^+ - d^-)$ is asymptotically optimal [29, 34]. We remark that in [21], it is shown how to combine the algorithms from [13] and [34] to achieve this bound without affecting the other properties shown in [13]; we are confident that the same technique can be applied to the algorithm proposed in this work. Finally, all our algorithms work with any clock drift parameter $1 < \vartheta \leq 1.007$, that is, the nodes' clocks can have up to 0.7% drift. In comparison, cheap quartz oscillators achieve $\vartheta \approx 1 + 10^{-5}$.

We consider our results of interest beyond the immediate improvements in complexity of the best known algorithms for pulse synchronisation. Since our framework may employ any consensus algorithm, it proves that pulse synchronisation is, essentially, *as easy* as synchronous consensus – a problem without the requirement for self-stabilisation or any timing uncertainty. Apart from the possibility for future improvements in consensus algorithms carrying over, this accentuates the following question:

> Is pulse synchronisation *at least as hard* as synchronous consensus?

Due to the various lower bounds and impossibility results on consensus [8, 10, 20, 30] mentioned earlier, a positive answer would immediately imply that the presented techniques are near-optimal. However, one may speculate that pulse synchronisation may rather have the character of (synchronous) approximate agreement [12, 17], as *precise* synchronisation of the pulse events at different nodes is not required. Considering that approximate agreement can be deterministically solved in $O(\log n)$ rounds, a negative answer is a clear possibility as well. Given that all currently known solutions either explicitly solve consensus, leverage techniques that are likely to be strong enough to solve consensus, or are very slow, this would suggest that new algorithmic techniques and insights into the problem are necessary.

1.4 Outline of this paper

Section 2 starts with an overview of the ingredients used in our construction. Section 3 establishes some key definitions and Section 4 presents the main results. As the formal treatment of our results is fairly involved, we start with an informal summary of the main ideas and techniques behind our results in Section 5. After this, we proceed to give all the details in the subsequent sections. Section 6 gives a detailed definition of the model of computing, Sections 7–9 give proofs of our technical results, and Section 10 concludes the paper by showing how to adapt our framework to utilise randomised consensus algorithms.

2 Ingredients of the framework

In this section, we give a high-level overview of our approach and compare it to previous solutions. We combine both novel and prior techniques in order to construct efficient pulse synchronisation algorithms. We take the following key steps:

(a) We start from a simple *non-self-stabilising* Byzantine-tolerant pulse synchronisation algorithm, which can be interpreted as a special case of the Srikanth–Toueg algorithm [33]. This algorithm operates in the bounded-delay model and relies only on simple timeouts and threshold voting mechanisms.

(b) We modify the above algorithm to explicitly employ a consensus routine (instead of threshold voting) to have nodes agree on whether a new pulse should be generated. In addition, we include a new "recovery" state, which nodes transition into if they observe inconsistent behaviour. Nodes in the recovery state refrain from generating pulses until all nodes agree to pulse again. This algorithm is not yet self-stabilising in itself.

(c) We design a self-stabilising *resynchronisation algorithm* that solves a weak variant of the pulse synchronisation problem. It guarantees that, *eventually,* within a small time window all correct nodes generate a resynchronisation pulse such that no node generates a new resynchronisation pulse for a long time. Unlike in the pulse synchronisation problem, this guarantee is limited to a *single* such pulse; otherwise, the behavior can be arbitrary.

(d) In order to make (b) self-stabilising, we devise an auxiliary stabilisation mechanism that acts as a wrapper for the consensus routine driving the algorithm. The auxiliary mechanism uses (a) to simulate any given synchronous consensus routine in the bounded-delay model. Furthermore, the resynchronisation pulses generated by (c) are used to initialise the stabilisation mechanism in a self-stabilising manner. Once a resynchronisation pulse occurs, the mechanism triggers a new stabilisation attempt. During the attempt, it is guaranteed that either (1) correct nodes quickly synchronise (or remain synchronised if already stabilised) or (2) all correct nodes observe inconsistent behaviour and end up in the recovery state of algorithm (b). In the latter case, the auxiliary stabilisation mechanism exploits the common timebase provided by the resynchronisation pulse to synchronise the nodes. Finally, if the system has already stabilised, then no subsequent stabilisation attempt triggered by a resynchronisation pulse interferes with the correct operation of the system.

While we build upon existing techniques, our approach has many key differences. First of all, while Dolev et al. [13] also utilise the concept of resynchronisation pulses, these are generated probabilistically. Moreover, their approach has an inherent time bound of $\Omega(f)$ for generating such pulses. In contrast, we devise a new recursive scheme that allows us to (1) *deterministically* generate resynchronisation pulses in $\Theta(f)$ time and (2) *probabilistically* generate resynchronisation pulses in $o(f)$ time.

To construct algorithms that generate resynchronisation pulses, we employ resilience boosting and filtering techniques inspired by our recent line of work on digital clock synchronisation in the *synchronous* model [25, 26, 28]. One of its main motivations was to gain a better understanding of the linear time/communication complexity barrier that research on pulse synchronisation ran into, without being distracted by the additional timing uncertainties due to communication delay and clock drift. The challenge here is to port these newly developed tools from the synchronous model to the bounded-delay bounded-drift model in a way that keeps them in working condition. Unsurprisingly, this is a non-trivial task. Moreover, while the approach in [25, 28] uses consensus, it does not directly lend itself to the use of arbitrary consensus routines, whereas [26] does not guarantee that stabilisation persists in the randomised setting (i.e., there is a positive probability that the algorithm fails after stabilisation). We resolve these issues in the bounded-delay model.

3 Preliminaries

We now establish some basic notation and definitions. Let V denote the set of all n nodes, $F \subseteq V$ be the set of faulty nodes such that $|F| < n/3$, and $G = V \setminus F$ the set of correct nodes. The sets G and F are unkown to the correct nodes in the system. We assume a continous reference time $[0, \infty)$ that is *not* available to the nodes in the distributed system. The reference time is only used to reason about the behaviour of the system. The adversary can choose the initial state of the system (memory contents, initial clock values, any messages in transit), the set F of faulty nodes which it controls, how the correct nodes' clocks progress and what is the delay of each individual message within the respective maximum clock drift and message delay bounds of ϑ and d. We assume that ϑ and d are known constants. The full formal description of the model is given in Appendix 6.

3.1 Pulse synchronisation algorithms

In the pulse synchronisation problem, the task is to have all the correct nodes locally generate pulse events in an almost synchronised fashion, despite arbitrary initial states and the presence of Byzantine faulty nodes. In addition, these pulses have to be well-separated. Let $p(v,t) \in \{0,1\}$ indicate whether a correct node $v \in G$ generates a pulse at time t. Moreover, let $p_k(v,t) \in [t,\infty)$ denote the time when node v generates the kth pulse event at or after time t and $p_k(v,t) = \infty$ if no such time exists. We say that the system has stabilised from time t onwards if

1. $p_1(v,t) \leq t + \Phi^+$ for all $v \in G$,
2. $|p_k(v,t) - p_k(u,t)| < \sigma$ for all $u, v \in G$ and $k \geq 1$,
3. $\Phi^- \leq p_{k+1}(v,t) - \min\{p_k(u,t) : u \in G\} \leq \Phi^+$ for all $v \in G$ and $k \geq 1$,

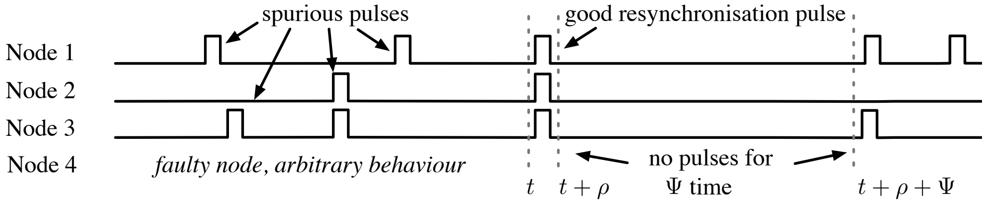
where Φ^- and Φ^+ are the accuracy bounds controlling the separation of the generated pulses. That is, (1) all correct nodes generate a pulse during the interval $[t, t + \Phi^+]$, (2) the kth pulse of any two correct nodes is less than σ time apart, and (3) for any pair of correct nodes their subsequent pulses are at least Φ^- but at most Φ^+ time apart.

We say that $\mathbf{A}$ is an f-resilient pulse synchronisation algorithm with *skew* σ and *accuracy* $\Phi = (\Phi^-, \Phi^+)$ with stabilisation time $T(\mathbf{A})$, if for any choices of the adversary such that $|F| \leq f$, there exists a time $t \leq T(\mathbf{A})$ such that the system stabilises from time t onwards. Moreover, a pulse synchronisation algorithm $\mathbf{A}$ is said to be a T-*pulser* if the accuracy bounds satisfy $\Phi^-, \Phi^+ \in \Theta(T)$. We use $M(\mathbf{A})$ to denote the maximum number of bits a correct node communicates per unit time when executing $\mathbf{A}$.

3.2 Resynchronisation algorithms

In our pulse synchronisation algorithm, we use so-called resynchronisation pulses to facilitate stabilisation. Essentially, the resynchronisation pulses are given by a weak variant of a pulse synchronisation algorithm, where the guarantee is that at some point all correct nodes generate a pulse almost synchronously, which is followed by a long period of silence. At all other times, the behaviour can be arbitrary.

Formally, we say that $\mathbf{B}$ is an f-resilient resynchronisation algorithm with skew ρ and separation window Ψ that stabilises in time $T(\mathbf{B})$ if the following holds: for any choices of the adversary such that $|F| \leq f$, there exists a time $t \leq T(\mathbf{B})$ such that every correct node $v \in G$ locally generates a *resynchronisation pulse* at time $r(v) \in [t, t+\rho)$ and no other resynchronisation pulse before time $t+\rho+\Psi$. We call such a resynchronisation pulse *good.* In particular, we do not impose any restrictions on what the nodes do outside the interval $[t, t+\rho+\Psi)$, that is, there may be *spurious* resynchronisation pulses outside this interval:



4 The transformation framework

Our main contribution is a modular framework that allows to turn any *non-self-stabilising* synchronous consensus algorithm into a self-stabilising pulse synchronisation algorithm in the bounded-delay model. In particular, this construction yields only a small overhead in time and communication complexity. This shows that efficient synchronous consensus algorithms imply efficient pulse synchronisation algorithms. As our construction is relatively involved, we opt to present it in a top-down fashion. In this section, we give our main result, which relies on two auxiliary results. We first state the main result, then state the two auxiliary results needed to obtain it, and finally show how these two results can be recursively applied to obtain new pulse synchronisation algorithms with increasing resilience. This approach is inspired by the resilience boosting techniques used for digital clock synchronisation algorithms in the synchronous model [25, 26, 28].

4.1 The main result

For notational convenience, we say that $\mathcal{C}$ is a *family of synchronous consensus routines* with running time $R(f)$ and message size $M(f)$, if for any $f \geq 0$ and $n \geq n(f)$, there exists a synchronous consensus algorithm $\mathbf{C} \in \mathcal{C}$ that runs correctly on n nodes given that there are at most f faulty nodes, terminates in $R(f)$ rounds, and uses messages of size $M(f)$. Here $n(f)$ gives the minimum number of nodes needed as a function of the resilience parameter f. Note that $R(f)$, $M(f)$, and $n(f)$ depend on $\mathcal{C}$; however, making this explicit would clutter notation. We emphasise that the algorithms in $\mathcal{C}$ are not assumed to be self-stabilising, that is, they work only if the system is properly initialised.

Our main technical result states that given a family of consensus routines, we can obtain pulse synchronisation algorithms with only small additional overhead.

Theorem 1. *Let $\mathcal{C}$ be a family of synchronous consensus routines that satisfy (i) for any $f_0, f_1 \in \mathbb{N}$, $n(f_0 + f_1) \leq n(f_0) + n(f_1)$ and (ii) both $M(f)$ and $R(f)$ are increasing. Then, for any $f \geq 0$, $n \geq n(f)$, and $1 < \vartheta \leq 1.007$, there exists a $T_0(f) \in \Theta(R(f))$, such that for any $T \geq T_0(f)$ we can construct a T-pulser $\mathbf{A}$ with skew $2d$. The stabilisation time $T(\mathbf{A})$ and number of bits $M(\mathbf{A})$ broadcasted per time unit satisfy*

$$T(\mathbf{A}) \in O\left(d + \sum_{k=0}^{\lceil \log f \rceil} R(2^k)\right) \quad \text{and} \quad M(\mathbf{A}) \in O\left(1 + \sum_{k=0}^{\lceil \log f \rceil} M(2^k)\right),$$

where the sums are empty when $f = 0$.

In the deterministic case, the *phase king algorithm* [5] provides a family of synchronous consensus routines that satisfy the requirements. Moreover, it achieves optimal resilience (i.e., the minimal possible $n(f) = 3f + 1$ [30]), constant message size, and asymptotically optimal [20] running time $R(f) \in O(f)$. Thus, this immediately yields the following result.

Corollary 1. *For any $f \geq 0$ and $n > 3f$, there exists a deterministic f-resilient pulse synchronisation algorithm over n nodes with skew $2d$ and accuracy bounds $\Phi^-, \Phi^+ \in \Theta(f)$ that stabilises in $O(f)$ time and has correct nodes broadcast $O(\log f)$ bits per time unit.*

4.2 The auxiliary results

The two main ingredients in the construction of Theorem 1 are a pulse synchronisation algorithm whose stabilisation mechanism is triggered by a resynchronisation pulse and a resynchronisation algorithm providing the latter. The result related to the first construction is stated below; the details of the construction are discussed later.

Theorem 2. *Let $f \geq 0$, $n > 3f$ and $(1 + \sqrt{5})/3 > \vartheta > 1$. Suppose for a network of n nodes there exist*

- *an f-resilient synchronous consensus algorithm $\mathbf{C}$, and*
- *an f-resilient resynchronisation algorithm $\mathbf{B}$ with skew $\rho \in O(d)$ and sufficiently large separation window $\Psi \in O(R)$ that tolerates clock drift of ϑ,*

where $\mathbf{C}$ runs in $R = R(f)$ rounds and lets nodes send at most $M = M(f)$ bits per round. Then a $\varphi_0(\vartheta) \in 1 + O(\vartheta - 1)$ exists so that for any constant $\varphi > \varphi_0(\vartheta)$ and sufficiently large $T \in O(R)$, there exists an f-resilient pulse synchronisation algorithm $\mathbf{A}$ for n nodes that

- *has skew $\sigma = 2d$ and satisfies the accuracy bounds $\Phi^- = T$ and $\Phi^+ = T\varphi$,*
- *stabilises in $T(\mathbf{B}) + O(R)$ time and has nodes broadcast $M(\mathbf{B}) + O(M)$ bits per time unit.*

To apply the above theorem, we require suitable consensus and resynchronisation algorithms. We rely on consensus algorithms from prior work and construct efficient resynchronisation algorithms ourselves. The idea is to combine pulse synchronisation algorithms that have *low resilience* to obtain resynchronisation algorithms with *high resilience.*

Theorem 3. *Let $f, n_0, n_1 \in \mathbb{N}$, $n = n_0 + n_1$, $f_0 = \lfloor (f-1)/2 \rfloor$, $f_1 = \lceil (f-1)/2 \rceil$, and $1 < \vartheta \leq 1.007$. Suppose that for some given $\Psi \in \Omega(1)$, sufficiently small constant $\varphi > \varphi_0(\vartheta)$, and $T_0 \in \Theta(\Psi)$, it holds that for any $h \in \{0, 1\}$ and $T_0 \leq T \in O(\Psi)$ there exists a pulse synchronisation algorithm $\mathbf{A}_h$ that*

- *runs on n_h nodes and tolerates f_h faulty nodes,*
- *has skew $\sigma = 2d$ and accuracy bounds $\Phi_h^- = T$ and $\Phi_h^+ = T\varphi$.*

Then there exists a resynchronisation algorithm with skew $\rho \in O(d)$ and separation window of length Ψ that generates a resynchronisation pulse by time $\max\{T(\mathbf{A}_0), T(\mathbf{A}_1)\} + O(\Psi)$, where nodes broadcast only $O(1)$ additional bits per time unit.

Thus, given a suitable consensus algorithm, one can now combine Theorems 2 and 3 to reduce the problem of constructing an f-resilient pulse synchronisation algorithm to finding algorithms that tolerate up to $\lfloor f/2 \rfloor$ faults and recurse; see Figure 1 for an overview.

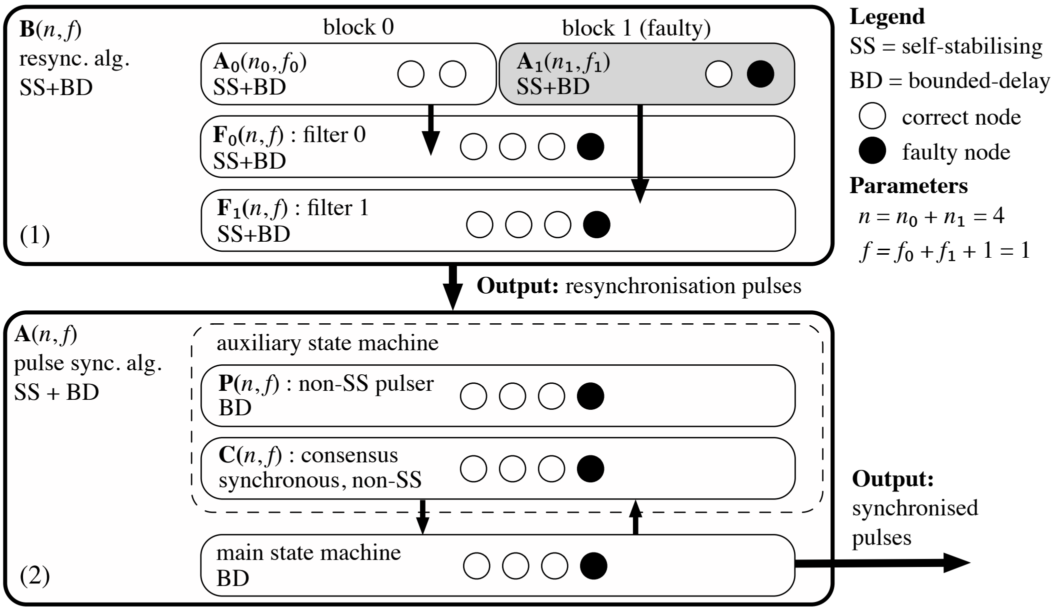


Figure 1: Ingredients for f-resilient resynchronisation and pulse synchronisation algorithms for n nodes, where $f = 1$ and $n = 4$. (1) The resynchronisation algorithm $\mathbf{B}$. The network is divided into two disjoint blocks, where each block $i \in \{0, 1\}$ runs a copy of a f_i-resilient pulse synchronisation algorithm $\mathbf{A}_i$ for n_i nodes. The output of $\mathbf{A}_i$ is fed into the filtering mechanism $\mathbf{F}_i$. The filtering mechanism $\mathbf{F}_i$ employs all the nodes in the network to impose some control on the (observed) behaviour of $\mathbf{A}_i$, because due to $f = f_0 + f_1 + 1$, (at most) one of the $\mathbf{A}_i$ is not guaranteed to stabilise. The grey block $i = 1$ contains more than $f_1 = 0$ faulty nodes, and hence, $\mathbf{A}_1$ may have arbitrary output. (2) The pulse synchronisation algorithm $\mathbf{A}$. The resynchronisation pulses from $\mathbf{B}$ are used to reset the stabilisation mechanism, which utilises two *non-self-stabilising* components: a pulse synchronisation algorithm $\mathbf{P}$ and a synchronous consensus routine $\mathbf{C}$, where the round-based algorithm $\mathbf{C}$ is simulated using $\mathbf{P}$. The stabilisation mechanism interacts with the main state machine to stabilise the output of $\mathbf{A}$ produced by the main state machine.

4.3 Proof of Theorem 1

Our proof takes an inductive approach. The idea is to interleave Theorem 2 and Theorem 3 to obtain pulse synchronisation algorithms with increasing resilience. First, we take a pulse synchronisation algorithms with small resilience. We then use this to devise a resynchronisation algorithm with higher resilience (Theorem 3). Second, we apply the resynchronisation algorithm together with a suitable consensus routine to obtain a new pulse synchronisation construction (Theorem 2). Thus, we obtain a pulse synchronisation algorithm with higher resilience, and can repeat the process.

Lemma 1. *Let $f, n_0, n_1 \in \mathbb{N}$, $n = n_0 + n_1$, $f_0 = \lfloor (f-1)/2 \rfloor$, and $f_1 = \lceil (f-1)/2 \rceil$. Suppose for $i \in \{0, 1\}$ there exists an f_i-resilient $\Theta(R)$-pulser $\mathbf{A}_i$ that runs on n_i nodes and whose accuracy bounds Φ_h^- and Φ_h^+ satisfy $\Phi_h^+ = \varphi \Phi_h^-$ for sufficiently small constants $\varphi > \vartheta$. Let $\mathbf{C}$ be an f-resilient consensus algorithm for a network of n nodes that has running time R and uses messages of at most M bits. Then there exists a $\Theta(R)$-pulser $\mathbf{A}$ that*

- *runs on n nodes and has resilience f,*
- *stabilises in time $T(\mathbf{A}) \in \max\{T(\mathbf{A}_0) + T(\mathbf{A}_1)\} + O(R)$,*
- *has nodes broadcast $M(\mathbf{A}) \in \max\{M(\mathbf{A}_0) + M(\mathbf{A}_1)\} + O(M)$ bits per time unit, and*
- *has skew $2d$ and whose accuracy bounds Φ^- and Φ^+ satisfy that $\Phi^+ = \varphi \Phi^-$.*

Proof. From Theorem 3, we get that for any sufficiently large $\Psi \in \Theta(R)$, there exists a resynchronisation algorithm $\mathbf{B}$ with skew $\rho \in O(d)$ and separation window of length Ψ that

- runs on n nodes and has resilience f,
- stabilises in time $\max\{T(\mathbf{A}_0) + T(\mathbf{A}_1)\} + O(\Psi) = \max\{T(\mathbf{A}_0) + T(\mathbf{A}_1)\} + O(R)$, and
- has nodes broadcast $\max\{M(\mathbf{A}_0) + M(\mathbf{A}_1)\} + O(1)$ bits per time unit.

We feed $\mathbf{B}$ and $\mathbf{C}$ into Theorem 2, yielding a pulse synchronisation algorithm $\mathbf{A}$ with the claimed properties, as the application of Theorem 2 increases the stabilisation time by an additional $O(R)$ time units and adds $O(M)$ bits per time unit. □

With these results in place, Theorem 1 is now a relatively straightforward consequence of the above lemma.

Remark 1. Note that one can convert an f-resilient pulse synchronisation algorithm $\mathbf{A}$ on n nodes into an f-resilient algorithm $\mathbf{A}'$ that runs on $n' > n$ nodes: (1) run $\mathbf{A}$ on the first n nodes and have them broadcast a bit when they generate a pulse locally and (2) have the remaining $n' - n$ nodes generate a pulse when they see at least $n - f > 2f + 1$ of the nodes running $\mathbf{A}$ generate a pulse.

Theorem 1. *Let $\mathcal{C}$ be a family of synchronous consensus routines that satisfy (i) for any $f_0, f_1 \in \mathbb{N}$, $n(f_0 + f_1) \leq n(f_0) + n(f_1)$ and (ii) both $M(f)$ and $R(f)$ are increasing. Then, for any $f \geq 0$, $n \geq n(f)$, and $1 < \vartheta \leq 1.007$, there exists a $T_0(f) \in \Theta(R(f))$, such that for any $T \geq T_0(f)$ we can construct a T-pulser $\mathbf{A}$ with skew $2d$. The stabilisation time $T(\mathbf{A})$ and number of bits $M(\mathbf{A})$ broadcasted per time unit satisfy*

$$T(\mathbf{A}) \in O\left(d + \sum_{k=0}^{\lceil \log f \rceil} R(2^k)\right) \quad \text{and} \quad M(\mathbf{A}) \in O\left(1 + \sum_{k=0}^{\lceil \log f \rceil} M(2^k)\right),$$

where the sums are empty when $f = 0$.

Proof. We prove the claim for $f \in \mathbb{N} = \bigcup_{k \in \mathbb{N}_0} [2^k, 2^{k+1})$ using induction on k. As base case, we use $f = 0$. This is trivial for all n: we pick a single node as a leader that generates a pulse when $\Phi^+ - \vartheta d$ units have passed on its local clock. Whenever the leader node pulses, all other nodes observe this within d time units. We have all other nodes generate a pulse whenever they observe the leader node generating a pulse. Thus, for $f = 0$ we have algorithms that stabilise in $O(d)$ time, broadcast $O(1)$ bits within d time, and have accuracy bounds of Φ^+ and $\Phi^- = \Phi^+/\vartheta - d$. Choosing $\Phi^+ \in O(1)$ sufficiently large, this results in $\Phi^+/\Phi^- \leq \varphi$ for any fixed constant $\varphi > \vartheta$.

For the inductive step, consider $f \in [2^k, 2^{k+1})$ and suppose that, for all $0 \leq f' < 2^k$ and $n' \geq n(f')$, there exists an f'-resilient pulser algorithm $\mathbf{A}'$ on n' nodes with accuracy bounds Φ' and $\varphi\Phi'$ for any $\Phi' \geq \Phi'_0 \in \Theta(R(f')$ and sufficiently small constant $\varphi > \vartheta$) so that

$$T(\mathbf{A}') \leq \alpha\left(d + \sum_{k'=0}^{\lceil \log f' \rceil} R(2^{k'})\right) \text{ and } M(\mathbf{A}') \leq \beta\left(1 + \sum_{k'=0}^{\lceil \log f' \rceil} M(2^{k'})\right),$$

where α and β are sufficiently large constants. In particular, we can now apply Lemma 1 with $f_0, f_1 \leq f/2 < 2^k$ and any $n \geq n(f)$, as $n(f) \geq n(f_0) + n(f_1)$ guarantees that we may choose some $n_0 \geq n(f_0)$ and $n_1 \geq n(f_1)$ such that $n = n_0 + n_1$. This yields an f-resilient $\Theta(R(f))$-pulser $\mathbf{A}$ over n nodes (with accuracy bounds Φ and $\varphi\Phi$ for any $\Phi \geq \Phi_0 \in \Theta(R(f))$), with stabilisation time

$$\begin{aligned} T(\mathbf{A}) &\leq \max\{T(\mathbf{A}_0) + T(\mathbf{A}_1)\} + \gamma R(f) \\ &\leq \alpha\left(d + \sum_{k=0}^{\lceil \log f/2 \rceil} R(2^k)\right) + \gamma R(2^{\lceil \log f \rceil}) \leq \alpha\left(d + \sum_{k=0}^{\lceil \log f \rceil} R(2^k)\right), \end{aligned}$$

where $\gamma \le \alpha$ is a constant; the second step uses that $R(f)$ is increasing. Similarly, the bound on the number of bits sent by $\mathbf{A}$ follows from Lemma 1 and the induction assumption:

$$\begin{aligned} M(\mathbf{A}) &\le \max\{M(\mathbf{A}_0) + M(\mathbf{A}_1)\} + \gamma' M(f) \\ &\le \beta \left(1 + \sum_{k=0}^{\lceil \log f/2 \rceil} M(2^k)\right) + \gamma' M(2^{\lceil \log f \rceil}) \le \beta \left(1 + \sum_{k=0}^{\lceil \log f \rceil} M(2^k)\right), \end{aligned}$$

where $\gamma' \le \beta$ is a constant and we used that $M(f)$ is increasing. □

4.4 Randomised algorithms

Employing randomised consensus algorithms in our framework is straightforward. We now summarise the main results related to randomised pulse synchronisation algorithms; the details are discussed later in Section 10. First, by applying our construction to a fast and communication-efficient randomised consensus algorithm, e.g. the one by King and Saia [22], we get an efficient randomised pulse synchronisation algorithm.

Corollary 2. *Suppose we have private channels. For any $f \ge 0$, constant $\varepsilon > 0$, and $n > (3+\varepsilon)f$, there exists a randomised f-resilient $\Theta(\operatorname{polylog} f)$-pulser over n nodes with skew $2d$ that stabilises in $\operatorname{polylog} f$ time w.h.p. and has nodes broadcast $\operatorname{polylog} f$ bits per time unit.*

We can also utilise the constant expected time protocol by Feldman and Micali [18]. With some care, we can show that for $R(f) \in O(1)$, Chernoff's bound readily implies that the stabilisation time is not only in $O(\log n)$ in expectation, but also with high probability.

Corollary 3. *Suppose we have private channels. For any $f \ge 0$ and $n > 3f$, there exists a randomised f-resilient $\Theta(\log f)$-pulser over n nodes with skew $2d$ that stabilises in $O(\log f)$ time w.h.p. and has nodes broadcast $\operatorname{polylog} n$ bits per time unit.*

5 Overview of techniques

In this section, we overview the elements used in our construction illustrated in Figure 1:

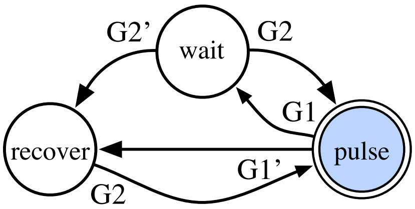
- a non-self-stabilising pulse synchronisation algorithm $\mathbf{P}$,
- a synchronous, non-self-stabilising consensus routine $\mathbf{C}$,
- a self-stabilising resynchronisation algorithm $\mathbf{B}$, and
- the constructed pulse synchronisation algorithm $\mathbf{A}$.

As the proofs are relatively involved due to a large number of technicalities arising from the uncertainties introduced by the clock drift and message delay, we now summarise the key ideas and deliberately skip over a number of details and avoid formalising the claims. Later in the subsequent sections, we will fill in all the missing details and give full proofs.

5.1 Our variant of the Srikanth–Toueg algorithm

The first component we need is a non-self-stabilising pulse synchronisation algorithm that tolerates Byzantine faults. To this end, we use a variant of the classic clock synchronisation algorithm by Srikanth and Toueg [33] that avoids transmitting clock values in favour of unlabelled pulses. As we do not require self-stabilisation for now, we can assume that all nodes receive an *initialisation signal* during the time window $[0, \tau)$ for a given parameter τ. The following theorem summarises the properties of the algorithm; the algorithm is given in Section 7.

Theorem 4. *Let $n > 1$, $f < n/3$, and $\tau > 0$. If every correct node receives an initialisation signal during $[0, \tau)$, then there exists a pulse synchronisation algorithm $\mathbf{P}$ such that:*



Transition guards:

G1 ≡ T_1 expires and saw $\geq n - f$ pulse messages within time T_1 at some point before the timeout expires

G1'≡ T_1 expires and G1 is not satisfied

G2 ≡ auxiliary machine signals output 1

G2'≡ T_{wait} expires or auxiliary machine signals output 0

Figure 2: The main state machine. When a node transitions to state PULSE, it generates a pulse event and sends a PULSE message to all nodes. When the node transitions to state WAIT, it broadcasts a WAIT message to all nodes. Guard G1 employs a sliding window memory buffer, which stores any PULSE messages that have arrived within time T_1. When a correct node transitions to PULSE, it resets a local timer of length T_1. Once it expires, either Guard G1 or Guard G1' become satisfied. Similarly, the timer T_{wait} is reset when a node transitions to WAIT. Once it expires, Guard G2' is satisfied and the node transitions from WAIT to RECOVER. The node transitions to state PULSE when Guard G2 is satisfied, which requires an "output 1" signal from the auxiliary state machine.

- *all correct nodes generate the first pulse (after initialisation) within time $O(\vartheta^2 d\tau)$,*
- *the pulses have skew $2d$,*
- *the accuracy bounds are $\Phi^- \in \Omega(\vartheta d)$ and $\Phi^+ \in O(\vartheta^2 d)$, and*
- *the algorithm communicates at most one bit per time unit.*

We can simulate synchronous message-passing algorithms with the above algorithm as follows. Assuming that no transient failures or new initialisation signals occur after time τ, by time $O(\vartheta^2 d\tau)$ the algorithm starts to generate pulses with skew $2d$ and accuracy bounds $\Phi^- \in \Omega(\vartheta d)$ and $\Phi^+ \in O(\vartheta^2 d)$. We can set the $\Omega(\vartheta d)$ term to be large enough so that all correct nodes can complete local computations and send/receive messages for each simulated round $i - 1$ before the ith pulse occurs. Thus, nodes can associate each message to a distinct round i (by counting locally) and simulate synchronous message-passing algorithms.

5.2 The self-stabilising pulse synchronisation algorithm (Theorem 2)

The general idea is to make the Srikanth–Toueg algorithm self-stabilising. Thus, we must deal with an arbitrary initial system state. In particular, the correct nodes may be scattered over the states, with inconsistent memory content, and also the timers employed in the transitions guards may have arbitrary values (within their domains). However, assume for the moment that there is a small window of length $\rho \in O(d)$ during which each node receives a *resynchronisation pulse*, which triggers the initialisation of the stabilisation mechanism.

The construction relies on two components:

1. a main state machine given in Figure 2, and
2. an auxiliary state machine that acts as a wrapper for an arbitrary consensus algorithm.

The main state machine is responsible for generating pulses, whereas the auxiliary state machine generates signals that drive the main state machine. The main machine works as follows: whenever a node enters the PULSE state, it waits for some time to see if at least $n - f$ nodes generated a pulse within a short time window. If not, the system has not stabilised, and the node goes into the RECOVER state to indicate this. Otherwise, the node goes into the WAIT state, where it remains for long enough to (a) separate any subsequent pulses from previous ones and (b) receive the next signal from the auxiliary machine. Once stabilised, the auxiliary machine is guaranteed to send the signal "1" within bounded time. This indicates that the node should pulse again. If no signal arrives on time or the signal is “0”, this means that the system has not stabilised and the node goes into the RECOVER state.

While the auxiliary state machine is slightly more involved, the basic idea is simple: (a) nodes try to check whether at least $n - f$ nodes transition to the WAIT state in the main state machine *in a short enough time window* (that is, time window which the nodes should follow during correct operation) and (b) then use a consensus routine to agree on this observation. Assuming that all correct nodes participate in the simulation of the consensus routine, we get the following:

- If the consensus algorithm **C** outputs “0”, then some $v \in G$ did not see $n - f$ nodes transitioning to WAIT in a short time window, and hence, the system has not yet stabilised.
- If the consensus algorithm **C** outputs “1”, then every $v \in G$ agrees that a transition to WAIT happened recently.

In particular, the idea is that when the system operates correctly, the consensus simulation will always succeed and output “1” at every correct node.

The obvious problem here is that the consensus routine is not self-stabilising and it operates in a synchronous model of computation. To remedy the latter problem, we use the algorithm from Theorem 4 to simulate round-based execution. However, this requires that an initialisation signal is generated within a time window of length τ, thus requiring some level of synchrony among the correct nodes. To wiggle our way out of this issue, we carefully construct the main state machine and auxiliary machine to satisfy the following properties:

1. The main state machine guarantees that if *some* correct node transitions to WAIT, then after a short interval no correct node transitions to WAIT for an extended period of time.

2. If a node $u \in G$ sees eat least $n - f$ nodes transitioning to WAIT in a short time window (including itself), then the node attempts to start a consensus instance with input “1”.

3. If node $u \in G$ attempts to start a simulation of consensus with input “1”, then at least $n - 2f > f$ correct nodes $v \in G$ must have recently transitioned to WAIT. As all nodes can reliably detect this event, this essentially ensures that their auxilliary machines synchronise. This way, we can guarantee that all correct nodes initialise a new consensus instance within τ time of each other and generate a consistent output.

4. If this output is “1”, all correct nodes generate a synchronised pulse and the system stabilises. Otherwise, all of them transition to state RECOVER.

5. If no $u \in G$ attempts to start a simulation of consensus with input “1” within a certain time, we make sure that all correct nodes end up in RECOVER. Here, we exploit that any consensus instance can be made *silent* [26], which means that no messages are sent by correct nodes if they all have input “0”. Hence, even if not all correct nodes actually participate in an instance, it does not matter as long as no correct node has input “1”.

Thus, either the system stabilises within a certain time or all correct nodes end up in state RECOVER. This is where we utilise the resynchronisation signals: when a resynchronisation signal is received, the nodes reset a local timer. Since the resynchronisation signal has a small skew of $\rho \in O(d)$, these timers expire within a relatively small time window as well. If the timer expires when all correct nodes are in the RECOVER state, then they can explicitly restart the system in synchrony, also resulting in stabilisation. The key here is to get a good resynchronisation pulse at some point, so that no spurious resynchronisation pulses interfere with the described stabilisation mechanism until it is complete. Once succesful, no correct nodes transition to RECOVER anymore. Thus, any subsequent resynchronisation pulses do not affect pulse generation. For a detailed discussion and formal analysis, see Appendix 8.

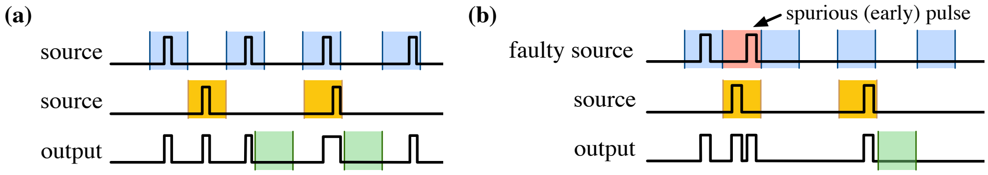


Figure 3: Idea of the resynchronisation algorithm. We take two pulse sources with coprime frequencies and output the logical OR of the two sources. In this example, the pulses of the first source should occur in the blue regions, whereas the pulses of the second source should hit the yellow regions. The green regions indicate a period where a pulse from either source is followed by at least Ψ time of silence. Eventually, such a region appears. (a) Two correct sources that pulse with set frequencies. (b) One faulty source that produces spurious pulses. Here, a pulse occurs too early (red region), and thus, we then enforce that the faulty source is silenced for $\Theta(\Psi)$ time.

5.3 Generating resynchronisation pulses (Theorem 3)

The final ingredient is a mechanism to generate resynchronisation pulses. We use a principle that has been previously used in digital clock synchronisation in the *synchronous* model [26]. However, as we now work in the bounded-delay model, we have to take into account the uncertainty created by clock drift. Nevertheless, while the technical details are much more involved, the underlying principle is the same.

Recall that a *good* resynchronisation pulse is an event triggered at all correct nodes within a small time interval, followed by at least Ψ time during which no correct node triggers a new such event. In order to construct an algorithm that generates such an event, we partition the set of n nodes into two disjoint *blocks* of roughly $n/2$ nodes. Each block runs an instance of a pulse synchronisation algorithm tolerating f_i faults, where $f_0 + f_1 + 1 = f$ (and $f_0 \approx f_1 \approx f/2$). For these two algorithms, we choose different pulsing frequencies (that is, accuracy bounds) that are roughly coprime integer multiples of the desired separation window Ψ. Both algorithms are used as potential sources of resynchronisation pulses. The idea behind our construction is illustrated in Figure 3. If both instances stabilise, it is not difficult to set up the frequencies such that $\mathbf{A}_i$ eventually generates a pulse that is not followed by a pulse from $\mathbf{A}_{1-i}$ within time Ψ.

Unfortunately, one of the instances (but not both) could have more than f_i faulty nodes, never stabilise, and thus generate possibly inconsistent pulses at arbitrary points in time. We overcome this by a two-step filtering process illustrated in Figure 4. First, we apply a number of threshold votes ensuring that if a pulse of a block is considered as a candidate resynchronisation pulse by *some* correct node, then *all* correct nodes observe this event. Second, we locally *filter out* any observed events that do not obey the prescribed frequency bounds for the respective block. Thus, the faulty block either generates (possibly inconsistent) pulses within the prescribed frequency bounds only, or its influence is suppressed entirely (for sufficiently long time). Either way, the correctly operating block will eventually succeed in generating a resynchronisation pulse.

5.4 Structure of the remainder of this paper

The rest of the paper is dedicated to detailed discussion, implementation, and analysis of the techniques discussed in this section. Section 6 starts with the formal definition of the model of computation, Section 7 described and analyses the non-self-stabilising pulse synchronisation algorithm, Sections 8 and 9 show Theorems 2 and 3, respectively. Finally in Section 10, we discuss the details of how to adapt the constructions to use randomised consensus routines.

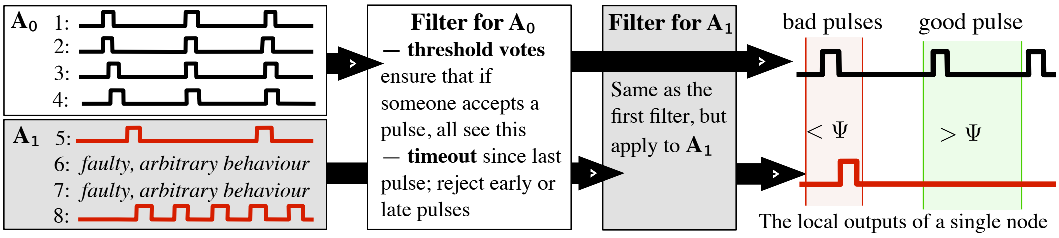

A0 1:
2:
3:
4:
A1 5:
6: faulty, arbitrary behaviour
7: faulty, arbitrary behaviour
8:
Filter for A0
— threshold votes ensure that if someone accepts a pulse, all see this
— timeout since last pulse; reject early or late pulses
Filter for A1
Same as the first filter, but apply to A1
bad pulses
good pulse
< Ψ
> Ψ
The local outputs of a single node

Figure 4: Example of the resynchronisation construction for 8 nodes tolerating 2 faults. We partition the network into two parts, each running a pulse synchronisation algorithm $\mathbf{A}_i$. The output of $\mathbf{A}_i$ is fed into the respective filter and any pulse that passes the filtering is used as a resynchronisation pulse. The filtering consists of (1) having *all* nodes in the network participate in a threshold vote to see if anyone thinks a pulse from $\mathbf{A}_i$ occurred (i.e. enough nodes running $\mathbf{A}_i$ generated a pulse) and (2) keeping track when was the last time a pulse from $\mathbf{A}_i$ occurred to check that the accuracy bounds of $\mathbf{A}_i$are respected: pulses that appear too early or too late are ignored.

6 The model of computation

In this section, we give a full formal definition of our model of computation and define notation used in the subsequent sections. Recall that the problem definitions were given in Section 3.

6.1 Reference time and clocks

Let $\mathbb{R}^+ = [0, \infty)$ denote the set of non-negative real numbers and $\mathbb{R}^+ \cup \{\infty\} = [0, \infty]$. Throughout this work, we assume a global *reference time* that is *not* available to the nodes in the distributed system. The reference time is only used to reason about the behaviour of the system. A *clock* is a strictly increasing function $C \colon \mathbb{R}^+ \to \mathbb{R}^+$ that maps the reference time to the local (perceived) time. That is, at time point t clock C indicates that the current time is $C(t)$. We say that a clock C has *drift* at most $\vartheta - 1 > 0$ if for any $t, t' \in \mathbb{R}^+$, where $t < t'$, the clock satisfies

$$t' - t \leq C(t') - C(t) \leq \vartheta(t' - t).$$

Throughout the paper, we assume that $\vartheta \in \Theta(1)$.

6.2 The communication model

We consider a *bounded-delay message-passing model* of distributed computation. The system is modelled as a fully-connected network of n nodes, where V denotes the set of all nodes. We assume that each node has a unique identifier from the set $[n] = \{0, 1, \dots, n-1\}$. Each node $v \in V$ has local clock $C(v) \colon \mathbb{R}^+ \to \mathbb{R}^+$ with maximum drift $\vartheta - 1$ for a global constant $\vartheta > 1$. We assume that the nodes cannot directly read their local clock values, but instead they can setup local timeouts of predetermined length. That is, a node can request to be signalled after t time units have passed on the node's *own local clock*.

For communication, we assume sender authentication, that is, each node can distinguish the sender of the messages it receives. In other words, every incoming communication link is labelled with the identifier of the sender. Unlike in fully-synchronous models, where communication and computation proceeds in lock-step at all nodes, we consider a model in which each message has an associated delay in $(0, d)$. For simplicity, we assume that the *maximum delay* $d \in \Theta(1)$ is a known constant, i.e., essentially we measure time in units of d. We assume that messages arrive in the order they are sent, i.e., the nodes communicate via first-in, first-out channels. We note that even though we assume continous, real-valued clocks, any constant offset in clock readings, e.g. due to discrete clocks, can be modelled by increasing d if needed.

6.3 Configurations and executions

An algorithm is a tuple $\mathbf{A} = (\mathcal{S}, \mathcal{T}, \mathcal{M}, \mathcal{P}, g, m)$, where $\mathcal{S}$ is a finite set of *states*, $\mathcal{T} \subseteq \mathbb{R}^+ \times 2^{\mathcal{S}}$ is a finite set of h *timers*, $g\colon [n] \times \mathcal{S} \times \mathcal{M}^n \times \{0,1\}^h \to \mathcal{S}$ is the state transition function, $m\colon \mathcal{S} \to \mathcal{M}$ is a message function, and $\mathcal{P} \subseteq \mathcal{S}$ defines the set of states that trigger a *pulse event*.

Local configuration. The local configuration $x(v,t)$ of a node v at time t consists of its own state $s(v,t) \in \mathcal{S}$, the state of its *input channels* $m(v,t) \in \mathcal{M}^n$, its local clock value $C(v,t) \in \mathbb{R}^+$, its *memory flags* $M(v,m,t) \subseteq [n]$, and timer states $T_k(v,t) \in [0,T_k]$ for each $(T_k, S) \in \mathcal{T}$ and $k \in [h]$. For convenience, we define *expiration flags* $e_k(v,t) \in \{0,1\}$, where $e_k(v,t) = 1 \Leftrightarrow T_k(v,t) = 0$, i.e., $e_k(v,t)$ is a shorthand indicating whether T_k is *expired* at time t at node v.

We assume that the transient faults have ceased at time 0, but the system is left in an *arbitrary initial state*. This entails that the initial values of the local clock $C(v,0)$, input channels $m(v,0)$, memory flags $M(v,m,0)$, and timer states $T_k(v,0)$ (and thus expiration flags) are arbitrary. Moreover, we assume that some of the nodes in the system may be *faulty*, that is, they have arbitrary (mis)behaviour and do not necessarily follow the given protocol $\mathbf{A}$. In the following, we use $F \subseteq [n]$ to denote an arbitrary set of *faulty nodes* and $G = [n] \setminus F$ the set of *correct* nodes. Once the algorithm has been fixed, the adversary can choose the set F as it pleases.

If at time t the value of either $m(v,t)$ or $e(v,t)$ changes, that is, node $v \in G$ receives a message or one of its local timers expires, the node sets its new local state to $s = g(v, s(v,t), m(v,t), e(v,t))$, where $s(v,t)$ is the node's previous state. In case $s(v,t) \neq s$, then we say that node v transitions to state s at time t. For convenicence, let us define the predicate $\Delta(v,s,t) = 1$ if $v \in G$ transitions to s at time t and $\Delta(v,s,t) = 0$ otherwise.

Local timers and timeouts. Each correct node $v \in G$ has a timer $(T_k, S) \in \mathcal{T}$, where T_k gives the duration of the timer and $S \subseteq \mathcal{S}$ is the set of states that reset the timer. The idea is that when node v transitions to state $s \in S$, node v resets all timers associated with state s. For all $t > 0$, we define the timer state as

$$T_k(v,t) = \max\{0, T_k(v, t_{\text{reset}}) - (C(v,t) - C(v, t_{\text{reset}})\},$$

where t_{reset} is the last time node v reset the timer T_k or time 0, that is,

$$t_{\text{reset}} = \max(\{0\} \cup \{t' \leq t : \Delta(v,s,t') = 1, s \in S\}).$$

Note that $T_k(v, t_{\text{reset}}) = T_k$ unless $t_{\text{reset}} = 0$, since with the exception of the arbitrary initial states the timer state is reset to T_k at time t_{reset}. Thus, at all times the timer state $T_k(v,t) \in [0, T_k]$ indicates how much time needs to pass on the node's local clock $C(v,\cdot)$ until the timer expires.

Communication. Let the *communication delay function* $d_{uv}\colon \mathbb{R}^+ \to \mathbb{R}^+$ be a strictly increasing function such that $0 < d_{uv}(t) - t < d$. The input channels of node $u \in G$ satisfy

$$m_v(u, d_{uv}(t)) = \begin{cases} m(s(v,t)) & \text{if } v \in G \\ b(u,v,t) & \text{otherwise,} \end{cases}$$

where $b(u,v,t) \in \mathcal{M}$ is the message a faulty node $v \in F$ decides to send to a correct node $u \in G$ at time t. We assume the adversary can freely choose the communication delay functions d_{uv}. Thus, the adversary can control what correct nodes receive from faulty nodes *and* how long the messages sent by correct nodes traverse (up to the maximum delay bound d). Intuitively, $m_v(u,t) \in \mathcal{M}$ denotes what was the *last* message received by node v from node u at time t. Again due to transient failures, we assume that $m_v(u,t) \in \mathcal{M}$ is arbitrary $t < d_{uv}(0)$.

Adversaries and executions. After fixing $f, n \in \mathbb{N}$ and an algorithm $\mathbf{A}$, we assume that the adversary chooses (1) the set $F \subseteq [n]$ of faulty nodes such that $|F| \leq f$, (2) the initial configuration $x(v, 0)$ for each $v \in G$, (3) the local clock values $C(v, t)$, (4) the message delays $d_{uv}(t)$, and (5) the messages $b(u, v, t)$ sent by faulty nodes for all $t \geq 0$.

Note that if the algorithm $\mathbf{A}$ is deterministic, then the adversary's choices for (1)–(5) together with $\mathbf{A}$ determine the execution, that is, local configurations $x(v, t)$ for all $v \in G$ and $t \geq 0$. Randomisation may be used in black-box calls to a consensus subroutine only. For the sake of a straightforward presentation, we therefore postpone the discussion of randomisation to Appendix 10, which covers the results obtained by utilising randomised consensus routines.

Memory flags. Nodes use *memory flags* to store which messages they have received. For a message of type $m \in \mathcal{M}$, we denote by $H(v, m, t) \subseteq [n]$ the nodes for which node v has enabled the flag for message m at time t. Similarly to resetting the local timers, a node can choose to clear the memory flags when transitioning to some prespecified states. Thus, if v clears the memory flags for message m at time t, then we have $H(v, m, t) = \emptyset$.

In our algorithms and their analysis, we are only interested in *how many* messages of a certain type have been observed. Hence, we use $|H(v, m, t)|$ gives the number of type m messages memorised by node v at time t. Finally, we write $K(v, m, t) = H(v, m, t) \cap G$ to denote the correct nodes that send a message m; this is only used for the analysis of the algorithms, as the nodes do not know the set of correct nodes.

Sliding windows and timed buffers. We often resort to the use of *sliding windows* in our transition guards instead of checking how many flags in total have been stored. A sliding window of length T is essentially a T-buffer that remembers from which nodes a certain message has been received within time T on the nodes local clock. Implementing such behaviour with timeouts and additional states is straightforward.

The behaviour of the timed buffers are as follows. Since the state of communication channels and timed buffers might be arbitrary at the initial state at time 0, we have that by time $T + d$ the contents of the timed buffer are guaranteed to be valid: if the buffer of $v \in G$ contains a message m from $u \in G$ at time $t \geq T + d$, then u must have sent an m message to v during the interval $(t - T - d, t)$ of reference time. Vice versa, if u sends a message m at time t to v, the buffer is guaranteed to contain the message during the interval $(t + d, t + T/\vartheta)$ of reference time.

Logical state machines. We will mostly describe our algorithms using relatively simple state machines, where state transitions are conditioned on the expiration flags and the number of set memory flags. In these state machines, instead of representing each state $s \in \mathcal{S}$ explicitly, we define a set $\mathcal{X}$ of *logical states* and identify each state $s \in \mathcal{S}$ to some logical state $\ell(s) \in \mathcal{X}$. That is, we have a surjective projection $\ell \colon \mathcal{S} \to \mathcal{X}$ that maps each state s onto its equivalence class $\ell(s)$, i.e., the logical state. For example, the algorithm in Appendix 7 uses the logical states $\mathcal{X} = \{\textsc{reset}, \textsc{start}, \textsc{ready}, \textsc{propose}, \textsc{pulse}\}$ and the transitions between these states are conditioned on the number of set memory flags and expiration flags. In addition, all timers will be associated to some logical state, that is, for every $(T_k, S) \in \mathcal{T}$ we have that $S \in \ell^{-1}(\mathcal{X})$ is an equivalence class of states. The logical states are being heavily utilised in the presentation of the algorithm in Appendix 8.

7 Byzantine-tolerant pulse synchronisation

We now describe the non-self-stabilising pulse synchronisation algorithm we utilise later in our construction of the self-stabilising algorithm. The non-self-stabilising algorithm is a variant of the Byzantine fault-tolerant clock synchronisation algorithm by Srikanth and Toeug [33] that avoids transmitting clock values in favor of unlabelled pulses.

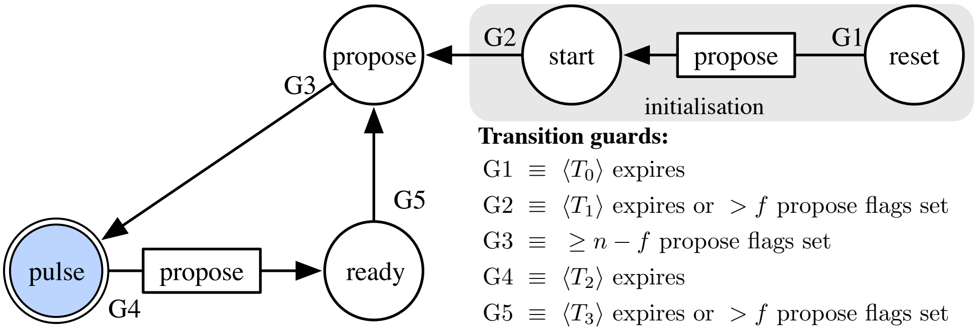

G2
start
propose
G1
reset
propose
G3
initialisation
Transition guards:
G1 ≡ ⟨T_0⟩ expires
G2 ≡ ⟨T_1⟩ expires or > f propose flags set
G3 ≡ ≥ $n-f$ propose flags set
G4 ≡ ⟨T_2⟩ expires
G5 ≡ ⟨T_3⟩ expires or > f propose flags set
G5
pulse
propose
ready
G4

Figure 5: The state machine for the non-self-stabilising pulse synchronisation algorithm. State transitions occur when the condition next to the edge is satisfied. All transition guards involve checking whether a local timer expires or a node has received sufficiently many messages from nodes in state PROPOSE. The only communication that occurs is when a node transitions to state PROPOSE; when this happens a node broadcasts this information to all others. The notation $\langle T \rangle$ indicates that T time units have passed on the local clock since the transition to the current state. The box labelled "propose" indicates that a node clears its PROPOSE memory flags when transitioning from PULSE to READY. That is, the node forgets who they have "seen" in PROPOSE the previous iteration. The same holds true for the transition from RESET to START. The algorithm assumes that during the interval $[0, \tau)$ all nodes transition to RESET. This starts the initialisation phase of the algorithm. Eventually, all nodes transition to PULSE within a short time window and start executing the algorithm. Whenever a node transitions to state PULSE it generates a local pulse event. Table 2 lists the constraints imposed on the timeouts.

Instead of a synchronous start, we assume that all nodes receive an initialisation signal during the time window $[0, \tau)$. In other words, nodes can start executing the algorithm at different times, but they all do so by some bounded (possibly non-constant) time τ. When a node receives the initialisation signal, it immediately transitions to the RESET state, whose purpose is to consistently clear up local memory and wait for other nodes to start executing the algorithm. However, nodes may behave arbitrarily before they receive the initialisation signal.

Later, when we make use of the algorithm as a subroutine for a self-stabilising algorithm, we need to consider the possibility that there are still messages from earlier (possibly corrupted) instances in transit, or nodes may be executing a previous instance in an incorrect way. Given the initialisation signal, this is easily overcome by waiting for sufficient time before leaving the starting state: waiting $\vartheta(\tau + d) \in O(\tau)$ local time guarantees that (i) all correct nodes transitioned to the starting state and (ii) all messages sent before these transitions have arrived. Clearing memory when leaving the starting state thus ensures that no obsolete information from previous instances is stored by correct nodes.

The algorithm is illustrated in Figure 5. In the figure, the circles denote the basic logical states (RESET, START, READY, PROPOSE, PULSE) of the state machine for each node. The two states RESET and START are only used during the initialisation phase of the algorithm that occurs when all nodes receive the initialisation signal during $[0, \tau)$. In Figure 5, directed edges between the states denote possible state transitions and labels give the conditions (transition guards) when the transition is allowed to (and must) occur. The notation is as follows: $\langle T_k \rangle$ means that timer T_k has expired, that is, the expiration flag satisfies $e_k(v, t) = 1$. Here, it is assumed that a timer is associated with the logical state which is left when it is expired. The constraints we impose on the timeouts are given in Table 2. The expression "$> f$ propose propose flags set" denotes the condition $|H(v, \text{PROPOSE}, t)| > f$. For simplicity, we assume in all our descriptions that every time a node transitions to a logical state, it broadcasts the name of the state to all other nodes. Given that we use a constant number of logical states per node (and in our

(1)	$T_0/\vartheta \geq \tau + d$
(2)	$T_1/\vartheta \geq (1 - 1/\vartheta)T_0 + \tau$
(3)	$T_2/\vartheta \geq 3d$
(4)	$T_3/\vartheta \geq (1 - 1/\vartheta)T_2 + 2d$

Table 2: The list of conditions used in the non-self-stabilising pulse synchronisation algorithm given in Figure 5. Recall that $d, \vartheta \in O(1)$ and τ is a parameter of the algorithm.

algorithms nodes can only undergo a constant number of state transitions in constant time), this requires $O(1)$ bits per time unit. In fact, closer inspection reveals that 1 bit per iteration of the cycle suffices here: the only relevant information is whether a node is in state PROPOSE or not.

The boxes labelled with "propose" indicate that when a node transitions to the designated state, it clears its memory flags for message PROPOSE. For example, if node v transitions to state READY at time t, then it explicitly clears its propose flags and it holds that $K(v, \text{READY}, t) = \emptyset$.

The algorithm relies heavily on the property that there are at most $f < n/3$ faulty nodes. This allows the use of the following "vote-and-pull" technique. If some correct node receives a PROPOSE message from at least $n - f$ different nodes at time t, then we must have that at least $n - 2f > f$ of these originated from correct nodes during the interval $(t - d, t)$, as every message has a positive delay of less than d. Furthermore, it follows that before time $t + d$ *all* correct nodes receive more than f PROPOSE messages.

In particular, this "vote-and-pull" technique is used in the transition to states PROPOSE and PULSE. Suppose at some point all nodes are in READY. If some node transitions to PULSE, then it must have observed at least $n - f$ nodes in PROPOSE by Guard G3. This in turn implies that more than f correct nodes have transitioned to PROPOSE. This in turn will (in short time) "pull" nodes that still remain in either state READY into state PROPOSE. Thus, Guard G3 will eventually be satisfied at the all nodes. The same technique is also used in the transition from START to PROPOSE during the initialisation phase of the algorithm. More formally, we have the following property.

Remark 2. Suppose $f < n/3$. Let $u, v \in G$, $t \geq d$ and $I = (t - d, t + d)$. If $|H(v, t, \text{PROPOSE})| \geq n - f$, then $|K(u, t', \text{PROPOSE})| > f$ for some $t' \in I$ assuming u does not clear its PROPOSE flags during the interval I.

For all $v \in G$ and $t > 0$, let $p(v, t) \in \{0, 1\}$ indicate whether v transitions to state PULSE at time t. That is, we have $p(v, t) = 1$ if node $v \in G$ transitions to state PULSE at time t and $p(v, t) = 0$ otherwise.

Lemma 2. *There exists $t_0 < \tau + T_0 + T_1 + d$ such that for all $v \in G$ it holds that $p(v, t) = 1$ for $t \in [t_0, t_0 + 2d)$.*

Proof. Let $v \in G$. Node v receives the initialisation signal during some time $t_{\text{reset}}(v) \in [0, \tau)$ and transitions to state RESET. From RESET node transitions to START at some time $t_{\text{start}}(v) \in [t_{\text{reset}}(v) + T_0/\vartheta, t_{\text{reset}}(v) + T_0]$ when the timer T_0 in Guard G1 expires. Since $T_0/\vartheta \geq \tau + d$ by Constraint (1), we get that $t_{\text{start}}(v) \geq \tau + d$. Thus, for all $u, v \in G$ we have $t_{\text{reset}}(u) + d \leq t_{\text{start}}(v)$.

Moreover, v transitions to PROPOSE at some time $t_{\text{propose}}(v) \in [t_{\text{start}}(v), t_{\text{start}}(v) + T_1]$ when Guard G2 is satisfied. Hence, any $v \in G$ transitions to state PROPOSE by latest at time $t_{\text{start}}(v) + T_1 \leq t_{\text{reset}}(v) + T_0 + T_1 \leq \tau + T_0 + T_1$. Let $t_{\text{propose}} \geq \tau + d$ be the minimal time some node $v \in G$ transitions to state PROPOSE after transitioning to RESET during $[0, \tau)$. From our earlier argument, we get that for any $t \in [t_{\text{start}}(v), t_{\text{propose}}) \subseteq [\tau + d, t_{\text{propose}})$ node v satisfies $K(v, \text{PROPOSE}, t) = \emptyset$, that is, node v will not observe a PROPOSE message from any correct node $u \in G$ before time t_{propose}.

Note that $t_{\text{propose}} \in [(T_0 + T_1)/\vartheta, \tau + T_0 + T_1)$ by Guard G1 and Guard G2. By Constraint (2) and our previous bounds we have that $t_{\text{propose}} \geq T_0/\vartheta + (1 - 1/\vartheta)T_0 + \tau = T_0 + \tau \geq t_{\text{start}}(u)$ for any $u \in G$. Hence, after time $T_0 + \tau$, no $u \in G$ clears its PROPOSE flags before transitioning to PULSE at time $t_{\text{pulse}}(u)$. In particular, we now have that $t_{\text{propose}}(v) \leq \tau + T_0 + T_1$ and hence all nodes transition to PULSE by some time $t_{\text{pulse}}(v) < \tau + T_0 + T_1 + d$, as each $u \in G$ must have received a PROPOSE message by this time from least $n - f$ of correct nodes satisfying the condition of Guard G3.

Let $t_0 = \inf\{t_{\text{pulse}}(v) : v \in G\} < \tau + T_0 + T_1 + d$ be the minimal time some correct node transitions to state PULSE. It remains to argue that $t_{\text{pulse}}(v) \in [t_0, t_0 + 2d)$. Since some node $v \in G$ transitioned to PULSE at time t_0, we must have that its condition in Guard G3 was satisfied. That is, we have $|H(v, t_0, \text{PROPOSE})| \geq n - f$ implying that for any $u \in G$ we have $|K(u, t'_0, \text{PROPOSE})| > 0$ for some $t'_0 < t_0 + d$. Since $t_0 \geq t_{\text{propose}} \geq T_0 + \tau \geq t_{\text{start}}(u)$, we get that $t_{\text{propose}}(u) \leq t'_0 < t_0 + d$ for all $u \in G$. It now follows that at time $t''_0 < t'_0 + d < t_0 + 2d$ we have $|K(u, t''_0, \text{PROPOSE})| \geq n - f$ for all $u \in G$ implying that Guard G2 is satisfied for u. Thus, $t_{\text{pulse}}(u) \in [t_0, t''_0) \subseteq [t_0, t_0 + 2d)$. Finally, observe that we can satisfy the constraints in Table 2 by having each of the timers to be $O(\tau)$. Hence, $t_0 < \tau + T_0 + T_1 + d$. □

Let us now fix t_0 as given by the previous lemma. For every correct node $v \in G$, we define

$$p_0(v) = \inf\{t \geq t_0 : p(v, t) = 1\} \quad \text{and} \quad p_{i+1}(v) = p_{\text{next}}(v, p_i(v)),$$

where $p_{\text{next}}(v, t) = \inf\{t' > t : p(v, t') = 1\}$ is the next time after time t node v generates a pulse.

Lemma 3. *For all $i \geq 0$, there exist*

$$t_{i+1} \in [t_i + (T_2 + T_3)/\vartheta, t_i + T_2 + T_3 + 3d) \quad \textit{such that} \quad p_i(v) \in [t_i, t_i + 2d) \textit{ for all } v \in G.$$

Proof. We show the claim using induction on i. For the case $i = 0$, the claim $p_0(v) \in [t_0, t_0 + 2d)$ follows directly from Lemma 2. For the inductive step, suppose $p_i(v) \in [t_i, t_i + 2d)$ for all $v \in G$. Each $v \in G$ transitions to state READY at a time $t_{\text{ready}}(v) \in [t_i + T_2/\vartheta, t_i + 2d + T_2)$ by Guard G4. Moreover, by Constraint (4) we have that $t_{\text{ready}}(v) > t_i + T_2/\vartheta \geq t_i + 3d$. As no correct node transitions to PROPOSE during $[t_i + 2d, t_i + (T_2 + T_3)/\vartheta)$, this implies that no node receives a PROPOSE message from a correct node before time $t_{\text{propose}}(u)$ when some node u transitions to PROPOSE from READY. Observe that now $t_{\text{propose}}(u) > t_i + (T_2 + T_3)/\vartheta > t_i + 2d + T_2$ by Guard G5 and Constraint (4). We have $t_{\text{ready}}(v) < t_i + 2d + T_2 < t_{\text{propose}}(u)$ for all $u, v \in G$. Therefore, there exists a time $t_{\text{ready}} < t_i + 2d + T_2$ such that all correct nodes are in state READY and $|H(v, t_{\text{ready}}, \text{PROPOSE})| = 0$ for all $v \in G$.

Next observe that $t_{\text{propose}}(v) \leq t_i + 2d + T_2 + T_3$ for any $v \in G$. Hence, every $u \in G$ will receive a PROPOSE message from every $v \in G$ before time $t_{\text{propose}}(v) + d \leq t_i + 3d + T_2 + T_3$. Thus, by Guard G3 we have that u transitions to PULSE yielding that $p_{i+1}(v) \in [t_i + t_{\text{ready}}, t_i + 3d + T_2 + T_3) \subseteq [t_i + (T_2 + T_3)/\vartheta, t_i + T_2 + T_3 + 3d)$. Let $t_{i+1} = \inf\{p_{i+1}(v) : v \in G\}$. We have already established that $t_{i+1} \in [t_i + (T_2 + T_3)/\vartheta, t_i + T_2 + T_3 + 3d)$. Now using the same arguments as in Lemma 2 it follows that $t_{\text{propose}}(u) < t_{i+1} + d$ for any $u \in G$, as u must have received at least $n - 2f$ PROPOSE messages before time $t_{i+1} + d$ triggering the condition in Guard G5 for node u. Thus, Guard G3 will be satisfied before time $t_{i+1} + 2d$ implying that $p_{i+1}(u) \in [t_{i+1}, t_{i+1} + 2d)$ for any $u \in G$. □

Theorem 4. *Let $n > 1$, $f < n/3$, and $\tau > 0$. If every correct node receives an initialisation signal during $[0, \tau)$, then there exists a pulse synchronisation algorithm* **P** *such that:*

- *all correct nodes generate the first pulse (after initialisation) within time $O(\vartheta^2 d\tau)$,*
- *the pulses have skew $2d$,*
- *the accuracy bounds are $\Phi^- \in \Omega(\vartheta d)$ and $\Phi^+ \in O(\vartheta^2 d)$, and*
- *the algorithm communicates at most one bit per time unit.*

Proof. The constraints in Table 2 can be easily satisfied by setting $T_0 = \vartheta(\tau + d)$, $T_1 = \vartheta^2(1 - 1/\vartheta)(\tau + d) + \tau$, $T_2 = \vartheta 3d$ and $T_3 = \vartheta^2(1 - 1/\vartheta)3d + 2d$. By Lemma 2 we get that there exists $t_0 \in O(\vartheta^2 d\tau)$ such that all nodes generate the first pulse during the interval $[t_0, t_0 + 2d)$. Applying Lemma 3 we get that for all $i > 0$, we have that nodes generate the ith pulse during the interval $[t_i, t_i + 2d)$, where $t_i \in [t_{i-1} + (T_2 + T_3)/\vartheta, t_{i-1} + T_2 + T_3 + 3d) \subseteq [\Phi^-, \Phi^+)$. Note that $T_2 + T_3 \in \Theta(\vartheta^2 d)$ and $\vartheta, d \in O(1)$. These observations give the first two properties. For the final property, observe that nodes only need to communicate when they transition to PROPOSE. This follows by observing that node cannot transition to PROPOSE again before $T_2/\vartheta > 3d$ time has passed due to Guard G4. Thus, nodes need to communicate one bit every $\Omega(d)$ time. □

8 Self-stabilising pulse synchronisation

In this section, we show how to use a resynchronisation algorithm and a synchronous consensus routine to devise self-stabilising pulse synchronisation algorithms. More precisely, we establish the following result:

Theorem 2. *Let $f \geq 0$, $n > 3f$ and $(1 + \sqrt{5})/3 > \vartheta > 1$. Suppose for a network of n nodes there exist*

- *an f-resilient synchronous consensus algorithm $\mathbf{C}$, and*
- *an f-resilient resynchronisation algorithm $\mathbf{B}$ with skew $\rho \in O(d)$ and sufficiently large separation window $\Psi \in O(R)$ that tolerates clock drift of ϑ,*

where $\mathbf{C}$ runs in $R = R(f)$ rounds and lets nodes send at most $M = M(f)$ bits per round. Then a $\varphi_0(\vartheta) \in 1 + O(\vartheta - 1)$ exists so that for any constant $\varphi > \varphi_0(\vartheta)$ and sufficiently large $T \in O(R)$, there exists an f-resilient pulse synchronisation algorithm $\mathbf{A}$ for n nodes that

- *has skew $\sigma = 2d$ and satisfies the accuracy bounds $\Phi^- = T$ and $\Phi^+ = T\varphi$,*
- *stabilises in $T(\mathbf{B}) + O(R)$ time and has nodes broadcast $M(\mathbf{B}) + O(M)$ bits per time unit.*

8.1 Overview of the ingredients

The pulse synchronisation algorithm presented in this section consists of two state machines running in parallel:

1. the so-called main state machine that is responsible for pulse generation, and
2. an auxiliary machine, which assists in initiating consensus instances and stabilisation.

The main state machine indicates when pulses are generated and handles all the communication between nodes except for messages sent by simulated consensus instances. The latter are handled by the auxiliary state machine. The transitions in the main state machine are governed by a series of threshold votes, local timeouts, and signals from the auxiliary state machine.

As we are interested in *self-stabilising* algorithms, the main and auxiliary state machines may be arbitrarily initialised. To handle this, a stabilisation mechanism is used in conjunction to ensure that, regardless of the initial state of the system, all nodes eventually manage to synchronise their state machines. The stabilisation mechanism relies on the following three subroutines which are summarised in the latter part of Figure 1:

(a) a resynchronisation algorithm,
(b) the non-self-stabilising pulse synchronisation algorithm from Section 7,
(c) a silent, synchronous consensus algorithm.

Resynchronisation pulses. Recall that *resynchronisation algorithm* is a weak variant of a pulse synchronisation algorithm: it guarantees that *eventually*, within some bounded time T, all correct nodes generate a pulse such that no new pulse is generated for sufficiently long time Ψ. We call such pulses *resynchronisation pulses.* Note that at all other times, the algorithm may generate pulses at arbitrary frequencies and not necessarily at all correct nodes. Nevertheless, at some point all correct nodes are bound to generate a synchronised pulse. We leverage this property to cleanly re-initialise the stabilisation mechanism from time to time.

Throughout this section, we simply assume that during the time interval $[0, \rho)$ every correct node $v \in V \setminus F$ receives a single resynchronisation pulse. Moreover, we assume that no correct node receives a new resynchronisation pulse for at least Ψ time of this, where Ψ is sufficiently large value we determine later. In Section 9, we show how to devise efficient resynchronisation algorithms that produce such resynchronisation pulses.

Simulating synchronous consensus. The latter two subroutines (b) and (c) are used in conjunction as follows. We use our variant of the Srikanth–Toueg pulse synchronisation algorithm described in Section 7 to simulate a synchronous consensus algorithm (in the bounded-delay model). Note that while this pulse synchronisation algorithm is not self-stabilising, it works properly even if non-faulty nodes initialise the algorithm at different times as long as they do so within a time interval of length τ.

Assuming that the nodes initialise the non-self-stabilising pusle synchronisation algorithm within at most time τ apart, it is straightforward to simulate round-based (i.e., synchronous) algorithms: a pulse of the non-self-stabilising algorithm indicates that a new round of the synchronous algorithm can be started. By setting the delay between two pulses large enough, we can ensure that all nodes have (1) time to execute the local computations of the synchronous algorithm and (2) any messages related to a single round arrive at their destinations before a new pulse occurs.

Employing silent consensus. For convenience, we utilise so-called silent consensus routines. Silent consensus routines satisfy exactly the same properties as usual consensus routines (validity, agreement, and termination) with the addition that correct nodes send no messages in executions in which all nodes have input 0.

Definition 1 (Silent consensus)**.** *A consensus routine is* silent*, if in each execution in which all correct nodes have input* 0*, correct nodes send no messages.*

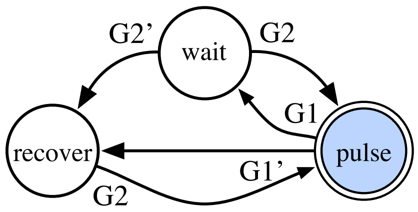
Any synchronous consensus routine can be converted into a silent consensus routine essentially for free. In our prior work [26], we showed that there exists a simple transformation that induces only an overhead of two rounds while keeping all other properties of the algorithm the same.

Theorem 5 ([26])**.** *Any consensus protocol* $\mathbf{C}$ *that runs in* R *rounds can be transformed into a silent consensus protocol* $\mathbf{C}'$ *that runs in* $R+2$ *rounds. Moreover, the resilience and message size of* $\mathbf{C}$ *and* $\mathbf{C}'$ *are the same.*

Thus, without loss of generality, we assume throughout this section that the given consensus routine $\mathbf{C}$ is silent. Moreover, this does not introduce any asymptotic loss in the running time or number of bits communicated.

8.2 High-level idea of the construction

The high-level strategy used in our construction is as follows. We run the resynchronisation algorithm in parallel to the self-stabilising pulse synchronisation algorithm we devise in this section. The resynchronisation algorithm will send the resynchronisation signals it generates to the pulse synchronisation algorithm.



Transition guards:

G1 ≡ T_1 expires and saw $\geq n - f$ pulse messages within time T_1 at some point before the timeout expires

G1'≡ T_1 expires and G1 is not satisfied

G2 ≡ auxiliary machine signals output 1

G2'≡ T_{wait} expires or auxiliary machine signals output 0

Figure 6: The main state machine. When a node transitions to PULSE state (blue circle) it will generate a local pulse event and send a PULSE message to all nodes. When the node transitions to WAIT state it broadcasts a WAIT message to all nodes. Guard G1 employs a sliding window memory buffer, which stores any PULSE messages that have arrived within time T_1. When a correct node transitions to PULSE it resets a local T_1 timeout. Once this expires, either Guard G1 or Guard G1' become satisfied. Similarly, the timer T_{wait} is reset when node transitions to WAIT. Once it expires, Guard G2' is satisfied and node transitions from WAIT to RECOVER. The node can transition to PULSE state when Guard G2 is satisfied, which requires an "output 1" signal from the auxiliary state machine given in Figure 7.

The pulse synchronisation algorithm consists of the main state machine given in Figure 6 and the auxiliary state machine given in Figure 7. The auxiliary state machine is responsible for generating the output signals that drive the main state machine (Guard G2 and Guard G2').

The auxiliary state machine employs a consensus routine to facilitate agreement among the nodes on whether a new pulse should occur. If the consensus simulation outputs 1, then the auxiliary state machine signals the main state machine to generate a pulse. Otherwise, if the consensus instance outputs 0, then this is used to signal that something is wrong and the system has not stabilised. We carefully set up our construction so that once the system stabilises, any consensus instance run by the nodes is guaranteed to always output 1 at every correct node.

As we operate under the assumption that the initial state of the system is arbitrary, the non-trivial part in our construction is to get all correct nodes synchronised well enough to even start simulating consensus jointly. This is where the resynchronisation algorithm comes into play. We make sure that the algorithm either stabilises or all nodes get "stuck" in a recovery state RECOVER. To deal with the latter case, we use the resynchronisation pulse to let all nodes synchronously reset a local timeout. Once this timeout expires, nodes that are in state RECOVER start a consensus instance with input "1". By the time this happens, either

- the algorithm has already stabilised (and thus no correct node is in state RECOVER), or
- all correct nodes are in state RECOVER and jointly start a consensus instance that will output "1" (by validity of the consensus routine).

In both cases, stabilisation is guaranteed.

Receiving a resynchronisation signal. The use of the resynchronisation signal is straightforward: when a correct node $u \in G$ receives a resynchronisation signal from the underlying resynchronisation algorithm $\mathbf{B}$, node u resets its local timeout T_{active} used by the auxiliary state machine in Figure 7. Upon expiration of the timeout, Guard G3 in the auxiliary state machine is activated only if the node is in state RECOVER at the time.

Main state machine. The main state machine, which is given in Figure 6, is responsible for generating the pulse events. It operates as follows. If a node is in state PULSE, it generates a local pulse event and sends a PULSE message to all other nodes. Now suppose a node $u \in G$ transitions to state PULSE. Two things can happen:

- If a node $u \in G$ is in state PULSE and observes at least $n - f$ nodes also generating a pulse within a short enough time window (Guard G1), it is possible that all correct nodes generated a pulse in a synchronised fashion. If this happens, then Guard G1 ensures that node u proceeds to the state WAIT. As the name suggests, the WAIT state is used to wait before generating a new pulse, ensuring that pulses obey the desired frequency bounds.

- Otherwise, if a node is certain that not all correct nodes are synchronised, it transitions to from PULSE to state RECOVER (Guard G1').

Once a node is in either WAIT or RECOVER, it will not leave the state before the consensus algorithm locally outputs "1", as Guard G2 needs to be satisfied in order for a transition to PULSE to take place. The simulation of consensus is handled by the auxiliary state machine, which we discuss below. The nodes use consensus to agree whether sufficiently many nodes transitioned to the WAIT state within a small enough time window. If the system has stabilised, all correct nodes transition to WAIT almost synchronously, and hence, after stabilisation every correct node always uses input "1" for the consensus instance.

Once a node transitions to state WAIT, it keeps track of how long it has been there. If the node observes that it has been there longer than it would take for a consensus simulation to complete under correct operation (proving that the system has not yet stabilised), it transitions to state RECOVER. Also, if the consensus instance outputs "0", the node knows something is wrong and transitions to RECOVER. During the stabilisation phase, nodes that transition to RECOVER refrain from using input "1" for any consensus routine before the local timeout T_{active} expires; we refer to the discussion of the auxiliary state machine.

Once the system stabilises, the behaviour of the main state machine is simple, as only Guard G1 and Guard G2 can be satisfied. This implies that correct nodes alternate between the PULSE and WAIT states. Under stabilised operation, we get that all correct nodes:

- transition to PULSE within a time window of length $2d$,

- observe that at least $n - f$ nodes transitioned to PULSE within a short enough time window ensuring that Guard G1 is satisfied at every correct node,

- transition to WAIT within a time window of length $O(d)$,

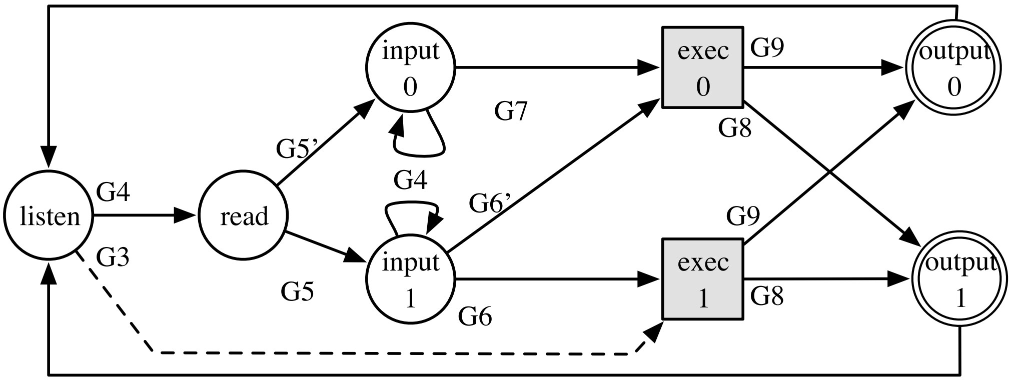
- correctly initialise a simulation of the consensus algorithm $\mathbf{C}$ with input "1", as correct nodes transitioned to WAIT in a synchronised fashion (see auxiliary state machine),

- all correct nodes remain in WAIT until Guard G2 or Guard G2' become satisfied.

Finally, we ensure that (after stabilisation) all correct nodes remain in state WAIT in the main state machine longer than it takes to properly initialise and simulate a consensus instance. This is achieved by using the T_{wait} timeout in Guard G2'. Due to the validity property of the consensus routine and the fact that all correct nodes use input 1, this entails that Guard G2 is always satisfied before Guard G2', such that all correct nodes again transition to PULSE within a time window of length $2d$.

Auxiliary state machine. The auxiliary state machine given in Figure 7 is slightly more involved. However, the basic idea is simple: (a) nodes try to check whether at least $n - f$ nodes transition to the WAIT state *in a short enough time window* (that is, a time window consistent with correct operation) and (b) then use a consensus routine to agree whether all nodes saw this. Assuming that all correct nodes participate in the simulation of consensus, we get the following:

- If the consensus algorithm $\mathbf{C}$ outputs "0", then some correct node did not see $n - f$ nodes transitioning to WAIT in a short time window, and hence, the system has not yet stabilised.



Transition guards:

G3 ≡ $\langle T_{\text{active}} \rangle$ expires while in recover

G4 ≡ $\geq f+1$ wait messages within time T_{listen}

G5 ≡ $\geq n-f$ wait messages within time T_{listen}

G5′ ≡ $\langle T_{\text{listen}} \rangle$ expires

G6 ≡ $\langle T_2 \rangle$ expires while not in recover

G6′ ≡ $\langle T_2 \rangle$ expires while in recover

G7 ≡ $\langle T_2 \rangle$ expires

G8 ≡ **A** outputs 1

G9 ≡ **A** outputs 0 or $\langle T_{\text{consensus}} \rangle$ expires or G4

Figure 7: The auxiliary state machine. The auxiliary state machine is responsible for initialising and simulating the consensus routine. The gray boxes denote states which represent the simulation of the consensus routine **C**. When transitioning to either EXEC 0 or EXEC 1, the node locally initialises the (non-self-stabilising) pulse synchronisation algorithm and a new instance of **C**. If the node transitions to EXEC 0, it uses input 0 for the consensus routine. If the node transitions to EXEC 1, it uses input 1, unless it is in state RECOVER when it locally starts the simulation. When the consensus simulation declares an output, the node transitions to either OUTPUT 0 or OUTPUT 1 (sending the respective output signal to the main state machine) and immediately into state LISTEN. The timeouts T_{listen}, T_2, and $T_{\text{consensus}}$ are reset when node transitions to the respective states that use a guard referring to them. The timeout T_{active} in Guard G3 (dashed line) is reset by the resynchronisation signal from the underlying resynchronisation algorithm **B**. Both INPUT 0 and INPUT 1 have a self-loop that is activated if Guard G4 is satisfied. This means that if Guard G4 is satisfied while in these states, the timer T_2 is reset.

- If the consensus algorithm **C** outputs “1”, then all correct nodes agree that a transition to WAIT happened recently.

In particular, the idea is that when the system operates correctly, the consensus simulation will always succeed and output “1” at every correct node.

The above idea is implemented in the auxiliary state machine as follows. Suppose that a correct node $u \in G$ is in the LISTEN state and the local timeout T_{active} is not about to expire (recall that T_{active} is only reset by the resynchronisation signal). Node u remains in this state until it is certain that at least one correct node transitions to WAIT in the main state machine. Once this happens, Guard G4 is satisfied and node u transitions to the READ state. In the READ state, node u waits for a while to see whether it observes (1) at least $n-f$ nodes transitioning to WAIT in a short time window or (2) less than $n-f$ nodes doing this.

In case (1), node u can be certain that at least $n-2f > f$ correct nodes transitioned to WAIT. Thus, node u can also be certain that every correct node observes at least $f+1$ correct nodes transitioning to WAIT; this will be a key property later. In case (2), node u can be certain that the system has not stabilised. If case (1) happens, we have that Guard G5 is eventually satisfied once a local time out of length $\Theta(d)$ expires. Once this occurs, node u transitions to

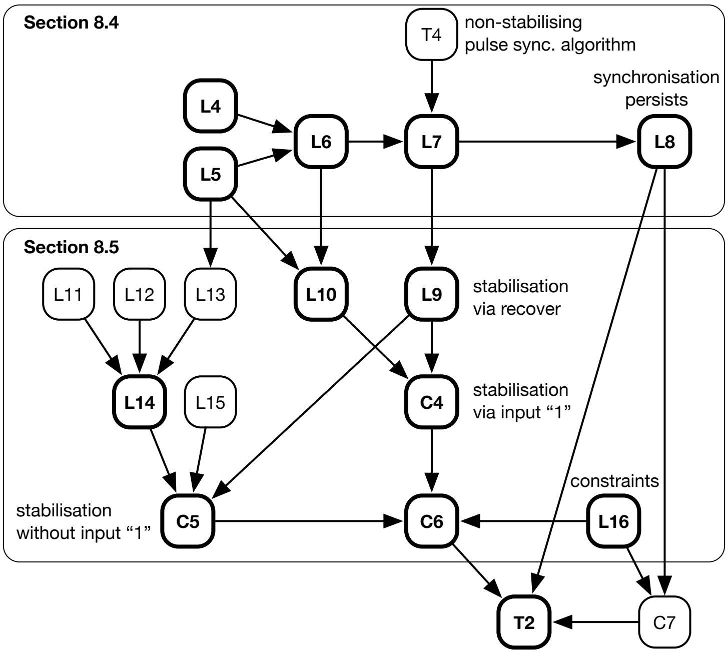


Figure 8: The overall structure of the proof. The bold rectangles denote results that are informally discussed in Section 8.3.

INPUT 1 indicating that node u is willing to use input "1" in the next simulation of consensus *unless* it is in the RECOVERstate in the main state machine. In case (2), we get that Guard G5' becomes satisfied and u transitions to INPUT 0. This means that u insists on using input "0" for the next consensus simulation.

Once node $u \in G$ transitions to either INPUT 0 or INPUT 1, it will remain there until the local timeout of length T_2 expires (see Guard G6, Guard G6' and Guard G7). However, if Guard G4 becomes satisfied *while* node u is in either of the input states, then the local timeout is reset again. We do this because if Guard G4 becomes satisfied while u is in one of the input states, (i) the same may be true for other correct nodes that are in state LISTEN and (ii) node u can be certain that the system has not stabilised. Resetting the timeout helps in ensuring that all correct nodes jointly start the next consensus instance (guaranteeing correct simulation), if Guard G4 is satisfied at all correct nodes at roughly the same time. In case this does not happen, resetting the timeout at least makes sure that there will be a time when *no* correct node is currently trying to simulate a consensus instance. These properties are critical for our proof of stabilisation.

8.3 Outline of the proof

We now give a high-level outline of our proof strategy. The structure of the proof is summarised in Figure 8. The key difficulty in achieving stabilisation is to ensure the proper simulation of a consensus routine despite the arbitrary initial state. In particular, after the transient faults cease, we might have some nodes attempting to execute consensus, whereas some do not. Moreover, nodes that are simulating consensus might be simulating different rounds of the consensus routine, and so on. To show that such disarray cannot last indefinitely long, we use the following arguments:

- if some correct node attempts to use input "1" for consensus, then at least $f + 1$ correct nodes have transitioned to WAIT in the main state machine (Lemma 4), that is, all correct nodes see if some other correct node might be initialising a new consensus instance with input “1” soon,

- if some correct node transitions to WAIT at time t, then there is a long interval of length $\Theta(T_2)$ during which no correct node transitions to WAIT (Lemma 5), that is, correct nodes cannot transition to WAIT state too often,

- if some correct node attempts to use input “1” for consensus, then all correct nodes initialise a new consensus instance within a time window of length $\tau \in \Theta((1 - 1/\vartheta)T_2)$ (Lemma 6),

- if all correct nodes initialise a new consensus instance within a time window of length τ, then all correct nodes participate in the same consensus instance and succesfully simulate an entire execution of **C** (Lemma 7).

The idea is that the timeout T_2 will be sufficiently large to ensure that consensus instances are well-separated: if a consensus instance is initialised with input “1” at some correct node, then there is enough time to properly simulate a complete execution of the consensus routine before any correct node attempts to start a new instance of consensus.

Once we have established the above properties, then it is easy to see that if synchronisation is established, then it *persists*. More specifically, we argue that if all correct nodes transition to PULSE at most time $2d$ apart, then all correct nodes initialise a new consensus instance within time τ using input “1” (Lemma 8). Thus, the system stabilises if all correct nodes eventually generate a pulse with skew at most $2d$.

Accordingly, a substantial part of the proof is arguing that all nodes eventually transition to PULSE within time window of $2d$. To see that this is bound to occur eventually, we consider an interval $[\alpha, \beta]$ of length $\Theta(R)$ and use the following line of reasoning:

- if all correct nodes are simultaneously in state RECOVER at some time before timeout T_{active} expires at any correct node, then Guard G3 in the auxiliary state machine becomes satisfied at all correct nodes and a new consensus instance with all-1 input is initialised within time τ (Lemma 9),

- if some correct node attempts to use input “1” within some time interval $[\alpha, \beta'] \subset [\alpha, \beta]$, then either (a) all correct nodes end up in RECOVER before timeout T_{active} expires at any node or (b) all correct nodes eventually transition to PULSE within time $2d$ (Lemma 10),

- if no correct node attempts to use input “1” during the time interval $[\alpha, \beta]$, all correct nodes will be in state RECOVER before the timeout T_{active} expires at any node (Lemma 14).

In either of the latter two cases, we can use the first argument to guarantee stabilisation (Corollary 4 and Corollary 5). Finally, we need to argue that all the timeouts employed in the construction can be set so that our arguments work out. The constraints related to all the timeouts are summarised in Table 3 and Lemma 16 shows that these can be satisfied. We now proceed to formalise and prove the above arguments in detail.

8.4 Analysing the state machines

First let us observe that after a short time, the arbitrary initial contents of the sliding window message buffers (implemented by simple state machines) have been cleared.

Remark 3. By time $t = \max\{T_1, T_{\text{listen}}\} + d \in O(\vartheta^2 d)$ the sliding window memory buffers used in Guard G2, Guard G4, and Guard G5 for each $u \in G$ are valid: if the buffer of Guard G2 contains a message m from $v \in G$ at any time $t' \geq t$, then $v \in G$ sent the message m during $(t - T_1 - d, t)$; similarly, for Guard G4 and Guard G5 this holds for the interval $(t - T_{\text{listen}} - d, t)$.

(5)	$d, \vartheta \in O(1)$
(6)	$T_1 = 3\vartheta d$
(7)	$T_{\text{listen}} = (\vartheta - 1)T_1 + 3\vartheta d$
(8)	$T_2 > \vartheta(T_{\text{listen}} + 3T_1 + 3d)$
(9)	$(2/\vartheta - 1)T_2 > T_{\text{listen}} + T_{\text{consensus}} + 5T_1 + 4d$
(10)	$\tau = \max\{(1 - 1/\vartheta)T_2 + T_{\text{listen}} + d + \max\{T_{\text{listen}} + d, 3T_1 + 2d\}, (1 - 1/\vartheta)T_{\text{active}} + \rho\}$
(11)	$T_{\text{consensus}} = \vartheta(\tau + T(R))$
(12)	$T_{\text{wait}} = T_2 + T_{\text{consensus}}$
(13)	$T_{\text{active}} \geq 4T_2 + T_{\text{listen}} + \vartheta(T_{\text{listen}} + T_{\text{wait}} - 5T_1 - 4d + \rho)$
(14)	$T_{\text{active}} \geq 2T_2 + T_{\text{consensus}} + \vartheta(3T_{\text{listen}} + 2T_{\text{wait}} + 3d + 2T_2 + 2T_{\text{consensus}})$

Table 3: The timeout conditions employed in the construction of Section 8.

Without loss of generality, we assume that this has happened by time 0. Moreover, we assume that every correct node received the resynchronisation signal during the time interval $[0, \rho)$. Thus, the memory contents of the message buffers are valid from time 0 on and every node has reset its T_{active} timeout during $[0, \rho)$. Hence, the timeout T_{active} expires at any node $u \in G$ during $[T_{\text{active}}/\vartheta, T_{\text{active}} + \rho)$.

We use $T(R) \in O(\vartheta^2 dR)$ to denote the maximum time a simulation of the R-round consensus routine $\mathbf{C}$ takes when employing the non-stabilising pulse synchronisation algorithm given in Section 7. We assume that the consensus routine $\mathbf{C}$ is silent, as by Theorem 5 we can convert any consensus routine into a silent one without any asymptotic loss in the running time.

First, we highlight some useful properties of the simulation scheme implemented in the auxiliary state machine.

Remark 4. If node $v \in G$ transitions to EXEC 0 or EXEC 1 at time t, then the following hold:

- node v remains in the respective state during $[t, t + \tau)$,
- node v does not execute the first round of $\mathbf{C}$ before time $t + \tau$,
- node v leaves the respective state and halts the simulation of $\mathbf{C}$ by time $[t + \tau, t + T_{\text{consensus}})$.

Now let us start by showing that if some node transitions to INPUT 1 in the auxiliary state machine, then there is a group of at least $f + 1$ correct nodes that transition to WAIT in the main state machine in rough synchrony.

Lemma 4. *Suppose node $v \in G$ transitions to* INPUT 1 *at time $t \geq T_{listen} + d$. Then there is a time $t' \in (t - T_{listen} - d, t)$ and a set $A \subseteq G$ with $|A| \geq f + 1$ such that each $w \in A$ transitions to* WAIT *during $[t', t]$.*

Proof. Since v transitions to INPUT 1, it must have observed at least $n - f$ distinct WAIT messages within time T_{listen} in order to satisfy Guard G5. As $f < n/3$, we have that at least $f + 1$ of these messages came from nodes $A \subseteq G$, where $|A| \geq f + 1$. The claim follows by choosing t' to be the minimal time during $(t - T_{\text{listen}} - d, t)$ at which some $w \in A$ transitioned to WAIT. □

Next we show that if some correct node transitions to WAIT, then it is followed by a long time interval during which no correct node transitions to WAIT. Thus, transitions to WAIT are well-separated.

Lemma 5. *Suppose node $v \in G$ transitions to* WAIT *at time $t \leq (T_{active} - T_2)/\vartheta$. Then no $u \in G$ transitions to* WAIT *during $[t + 3T_1 + d, t + T_2/\vartheta - 2T_1 - d)$.*

Proof. Since $v \in G$ transitioned to WAIT at time t, it must have seen at least $n - f$ nodes transitioning to PULSE during the time interval $(t - 2T_1, t)$. Since $f < n/3$, it follows that

$n - 2f \geq f + 1$ of these messages are from correct nodes. Let us denote this set of nodes by $A \subseteq G$.

Consider any node $w \in A$. As node w transitioned to PULSE during $(t - 2T_1 - d, t)$, it transitions to state RECOVER or to state WAIT at time $t_w \in (t - 2T_1 - d, t + T_1)$. Either way, as it also transitioned to LISTEN, transitioning to PULSE again requires to satisfy Guard G3 or one of Guard G6, Guard G6', and Guard G7 while being in states INPUT 0 or INPUT 1, respectively. By assumption, Guard G3 is not satisfied before time $T_{\text{active}}/\vartheta \geq t + T_2/\vartheta$, and the other options require a timeout of T_2 to expire, which takes at least time T_2/ϑ. It follows that w is not in state PULSE during $[t + T_1, t + T_2/\vartheta - 2T_1 - d)$.

We conclude that no $w \in A$ is observed transitioning to PULSE during $[t + T_1 + d, t + T_2/\vartheta - 2T_1 - d)$. Since $|A| > f$, we get that no $u \in G$ can activate Guard G1 and transition to WAIT during $[t + 3T_1 + d, t + T_2/\vartheta - 2T_1 - d)$, as $n - |A| < n - f$. □

Using the previous lemmas, we can show that if some correct node transitions to state INPUT 1 in the auxiliary state machine, then every correct node eventually initialises and participates in the same new consensus instance. That is, every correct node initialises the underlying Srikanth–Toueg pulse synchronisation algorithm within a time interval of length τ.

Lemma 6. *Suppose node $u \in G$ transitions to* INPUT 1 *at time $t \in [T_{listen} + d, (T_{active} - T_2)/\vartheta]$. Then each $v \in G$ transitions to state* EXEC 0 *or state* EXEC 1 *at time $t_v \in [t_0, t_0 + \tau)$, where $t_0 = t - T_{listen} - d + T_2/\vartheta$. Moreover, Guard G4 cannot be satisfied at any node $v \in G$ during $[t + 3T_1 + 2d, t^* + T_1/\vartheta)$, where $t^* := \min_{v \in G}\{p(v, t + d)\}$.*

Proof. By Lemma 4 there exists a set $A \subseteq G$ such that $|A| \geq f + 1$ and each $w \in A$ transition to WAIT at time $t_w \in (t - T_{\text{listen}} - d, t)$. This implies that Guard G4, and thus also Guard G9, becomes satisfied for $v \in G$ at time $t'_v \in [t - T_{\text{listen}} - d, t + d)$. Thus, every $v \in G$ transitions to state READ, INPUT 0, or INPUT 1 at time t'_v; note that if v was in state INPUT 0 or INPUT 1 before this happened, it transitions back to the same state due to Guard G4 being activated and resets its local T_2 timer. Moreover, by time $t'_v \leq r_v < t'_v + T_{\text{listen}} < t + d + T_{\text{listen}}$ node v transitions to either INPUT 0 or INPUT 1, as either Guard G5 or Guard G5' becomes activated in case v transitions to state READ at time t'_v.

Now we have that node v remains in either INPUT 1 or INPUT 0 for the interval $[r_v, r_v + T_2/\vartheta)$, as Guard G6, Guard G6', or Guard G7 are not satisfied before the local timer T_2 expires. Moreover, by applying Lemma 5 to any $w \in A$, we get that no $v \in G$ transitions to WAIT during the interval

$$[t_w + 3T_1 + d, t_w + T_2/\vartheta - 2T_1 - d) \supseteq (t + 3T_1 + d, t + T_2/\vartheta - 2T_1 - T_{\text{listen}} - 2d).$$

Recall that for each $v \in G$, $t'_v < t + d$. After this time, v cannot transition to PULSE again without transitioning to EXEC 0 or EXEC 1 first. Since $t + T_2/\vartheta - 2T_1 - T_{\text{listen}} - 2d > t + T_1 + d$ by Constraint (8), we get that every $w \in G$ has arrived in state WAIT or RECOVER by time $t + T_2/\vartheta - 2T_1 - T_{\text{listen}} - 2d$. Thus, no such node transitions to state WAIT during $[t + T_2/\vartheta - 2T_1 - T_{\text{listen}} - 2d, t^* + T_1/\vartheta)$: first, it must transition to PULSE, which requires to satisfy Guard G2, i.e., transitioning to state OUTPUT 1, and then a timeout of T_1 must expire; here, we use that we already observed that $t + T_2/\vartheta - 2T_1 - T_{\text{listen}} - 2d > t + d$, i.e., by definition the first node $w \in G$ to transition to PULSE after time $t + T_2/\vartheta - T_1 - T_{\text{listen}} - 2d$ does so at time t^*. We conclude that Guard G4 cannot be satisfied at any $v \in G$ during the interval $[t + 3T_1 + 2d, t^* + T_1/\vartheta)$, i.e., the second claim of the lemma holds.

We proceed to showing that each $v \in G$ transitions to state EXEC 0 or state EXEC 1 at time $t_v \in [t_0, t_0 + \tau)$, i.e., the first claim of the lemma. To this end, observe that w transitions to either state EXEC 0 or EXEC 1 at some time $t' \in (t'_w, t^*)$. By the above observations, $t' \geq r_w + T_2/\vartheta \geq t - T_{\text{listen}} - d + T_2/\vartheta = t_0$. Node w initialises the Srikanth-Toueg algorithm given in Figure 5 locally at time t'. In particular, by the properties of the simulation algorithm given in Remark 4, we have that w waits until time $t' + \tau$ before starting the simulation of $\mathbf{C}$, and hence, w remains in EXEC 0 or EXEC 1 at least until time $t' + \tau$ before the simulation of $\mathbf{C}$ produces an output value. Thus, we get that $t^* \geq t' + \tau \geq t_0 + \tau$.

Recall that each $v \in G$ resets timeout T_2 at time $r_v \in [t'_v, t + d + T_{\text{listen}}) \subseteq [t - T_{\text{listen}} - d, t + T_{\text{listen}} + d)$ and does not reset it during $[t + 3T_1 + 2d, t^* + T_1/\vartheta)$, as it does not satisfy Guard G4 at any time from this interval. When T_2 expires at v, it transitions to EXEC 0 or EXEC 1. Because $t^* + T_1/\vartheta > t_0 + \tau \geq t + \max\{T_{\text{listen}} + d, 3T_1 + 2d\} + T_2$ by Constraint (10), this happens at time $t_v \in [t - T_{\text{listen}} - d + T_2/\vartheta, t_0 + \tau] = [t_0, t_0 + \tau]$, as claimed. □

Next, we show that if all correct nodes initialise a new instance of the Srikanth–Toueg pulse synchronisation algorithm within a time interval of length τ, then every correct node initialises, participates in, and successfully completes simulation of the consensus routine $\mathbf{C}$.

Lemma 7. *Suppose there exists a time t_0 such that each node $v \in G$ transitions to* EXEC 0 *or* EXEC 1 *at some time $t_v \in [t_0, t_0 + \tau)$. Let t' be the minimal time larger than t_0 at which some $u \in G$ transitions to either* OUTPUT 0 *or* OUTPUT 1*. If Guard G4 is not satisfied at any $v \in G$ during $[t_0, t' + 2d)$, then $t' \leq t_0 + T_{consensus}/\vartheta - 2d$ and there are times $t'_v \in [t', t' + 2d)$, $v \in G$, such that:*

- *each $v \in G$ transitions to* OUTPUT 1 *or* OUTPUT 0 *at time t'_v (termination),*
- *this state is the same for each $v \in G$ (agreement), and*
- *if each $v \in G$ transitioned to state* EXEC 1 *at time t_v, then this state is* OUTPUT 1 *(validity).*

Proof. When $v \in G$ transitions to either EXEC 0 or EXEC 1, it sends an initialisation signal to the non-self-stabilising Srikanth–Toueg algorithm described in Section 7. Using the clock given by this algorithm, nodes simulate the consensus algorithm $\mathbf{C}$. If node $v \in G$ enters state EXEC 0, it uses input “0” for $\mathbf{C}$. Otherwise, if v enters EXEC 1 it uses input “1”.

Note that Theorem 4 implies that if all nodes initialise the Srikanth–Toueg algorithm within time τ apart, then the simulation of $\mathbf{C}$ takes at most $\tau + T(R) \in O(\vartheta^2 d(\tau + R))$ time. Moreover, all nodes will declare the output on the same round, and hence, declare the output within a time window of $2d$, as the skew of the pulses is at most $2d$.

Now let us consider the simulation taking place in the auxiliary state machine. If Guard G4 is not satisfied during $[t_0, t' + 2d)$ and the timer $T_{\text{consensus}}$ does not expire at any node $v \in G$ during the simulation, then by time $t' + 2d \leq t_0 + \tau + T(R) \in O(\vartheta^2 d(\tau + R))$ the nodes have simulated R rounds of $\mathbf{C}$ and declared output. By assumption, Guard G4 cannot be satisfied prior to time $t' + 2d$. At node $v \in G$, the timer $T_{\text{consensus}}$ is reset at time $t_v \geq t_0$. Hence, it cannot expire again earlier than time $t_0 + T_{\text{consensus}}/\vartheta \geq t_0 + \tau + T(R)$ by Constraint (11). Hence, the simulation succeeds.

Since the simulation of the consensus routine $\mathbf{C}$ completes at each $v \in G$ at some time $t'_v \in [t', t' + 2d)$, we get that Guard G8 or Guard G9 is satisfied at time t'_v at node v. Hence, v transitions to either of the output states depending on the output value of $\mathbf{C}$. The last two claims of the lemma follow from the agreement and validity properties of the consens routine $\mathbf{C}$. □

Now we can show that if the main state machine stabilises, it remains stabilised. More precisely, the following lemma shows that it is sufficient to have all correct nodes transition to PULSE within a time window of length $2d$: once this happens, all correct nodes remain synchronised with skew $2d$ and controlled accuracy bounds.

Lemma 8. *Suppose there exists an interval $[t, t + 2d)$ such that for all $v \in G$ it holds that $p(v, t_v) = 1$ for some $t_v \in [t, t + 2d)$. Then there exists $t' \in [t + T_2/\vartheta, t + (T_2 + T_{consensus})/\vartheta - 2d)$ such that $p_{next}(v, t_v) \in [t', t' + 2d)$ for all $v \in G$.*

Proof. First, observe that if any node $v \in G$ transitions to PULSE at time t_v, then node v will be in state LISTEN in the auxiliary state machine at time t_v. To see this, note that node v must have transitioned to OUTPUT 1 in the auxiliary state machine at time t_v in order to satisfy Guard G2 leading to state PULSE. Furthermore, once this happens, node v transitions immediately from OUTPUT 1 to LISTEN in the auxiliary state machine.

Next, note that $v \in G$ will not transition to RECOVER before time $t_v + T_1/\vartheta \geq t + 3d$ by Guard G1 and Constraint (6). By assumption, every $u \in G$ transitions to PULSE by time $t + 2d$, and thus, node v observes a PULSE message from at least $n - f$ correct nodes $u \in G$ by time $t + 3d$. Thus, all correct nodes observe a PULSE message from at least $n - f$ nodes during $[t, t + T_1/\vartheta) = [t, t + 3d)$ satisfying Guard G1. Hence, every correct node $v \in G$ transitions to WAIT during the interval $[t + T_1/\vartheta, t + T_1 + 2d)$ and remains there until Guard G2 or Guard G2' is activated.

Now let us consider the auxiliary state machine. From the above reasoning, we get that every node $v \in G$ will observe at least $n - f$ nodes transitioning from PULSE to WAIT during the interval $[t + T_1/\vartheta, t + T_1 + 3d)$. As we have $T_{\text{listen}}/\vartheta \geq (1 - 1/\vartheta)T_1 + 3d$ by Constraint (7), both Guard G4 and Guard G5 are satisfied for v during the same interval. Thus, node v transitions to state INPUT 1 during the interval $[t + T_1/\vartheta, t + T_1 + 3d)$. It also follows that v transitions to EXEC 1 during $[t + (T_1 + T_2)/\vartheta, t + T_1 + T_2 + 3d) \subseteq [t + T_2/\vartheta, t + T_2/\vartheta + \tau)$ by Constraint (10).

Note that $t + T_1 + 4d < t + T_2/\vartheta$ by Constraint (6) and Constraint (8) and that no correct node transitions to WAIT again after time $t + T_1 + 3d$ before transitioning to OUTPUT 1 and spending, by Constraint (6), at least $T_1/\vartheta > 2d$ time in PULSE. Therefore, we can apply Lemma 7 with $t_0 = t + T_2/\vartheta$, yielding a time $t' < t + T_2/\vartheta + T_{\text{consensus}}/\vartheta - 2d$ such that each $v \in G$ transitions to OUTPUT 1 in the auxiliary state machine at time $t'_v \in [t', t' + 2d)$. Hence, Guard G2 is satisfied at time t'_v. Note that Guard G2' cannot be satisfied: no correct node transitions to OUTPUT 0 and the timer T_{wait} expires after time $t + T_{\text{wait}}/\vartheta \geq t'$ by Constraint (12). □

8.5 Ensuring stabilisation

We showed above that if all correct nodes eventually generate a pulse with skew $2d$, then the pulse synchronisation routine stabilises. In this section, we show that this is bound to happen. The idea is that if stabilisation does not take place within a certain interval, then all nodes end up being simultaneously in state RECOVER in the main state machine and state LISTEN in the auxiliary state machine, until eventually timeout T_{active} expires. This in turn allows the "passive" stabilisation mechanism to activate by having timer T_{active} expire at every node and stabilise the system as shown by the lemma below.

Lemma 9. *Let $t < T_{active}/\vartheta$ and $t^* = \min_{v \in G}\{p_{next}(v, t)\}$. Suppose the following holds for every node $v \in G$:*

- *node v is in state* RECOVER *and* LISTEN *at time t, and*
- *Guard G4 is not satisfied at v during $[t, t^* + 2d)$,*

Then $t^* < T_{active} + \rho + T_{consensus}/\vartheta$ and every $v \in G$ transitions to PULSE *at time $t_v \in [t^*, t^* + 2d)$.*

Proof. Observe that Guard G3 is not satisfied before time $T_{\text{active}}/\vartheta$. As Guard G4 is not satisfied during $[t, t^*)$, no correct node leaves state T_{listen} before time $T_{\text{active}}/\vartheta$. Since no $v \in G$ can activate Guard G2 during $[t, t^*)$, we have that every $v \in G$ remains in state RECOVER during this interval. Let $t_0 \in [T_{\text{active}}/\vartheta, T_{\text{active}} + \rho)$ be the minimal time (after time t) when Guard G3 becomes satisfied at some node $v \in G$. From the properties of the simulation algorithm given in Remark 4, we get that no $w \in G$ transitions away from EXEC 1 before time $t_0 + \tau \geq T_{\text{active}}/\vartheta + \tau$.

Since $\tau \geq (1 - 1/\vartheta)T_{\text{active}} + \rho$ by Constraint (10), we conclude that no $w \in G$ transitions to OUTPUT 1 (and thus PULSE) before time $T_{\text{active}} + \rho$. Therefore, each $w \in G$ transitions to EXEC 1 at some time $r_w \in [T_{\text{active}}/\vartheta, T_{\text{active}} + \rho) \subseteq [t_0, t_0 + \tau)$. Recall that Guard G4 is not satisfied during $[t_0, t^* + 2d)$. Hence, we can apply Lemma 7 to the interval $[t_0, t_0 + \tau)$, implying that every $w \in G$ transitions to OUTPUT 1 at some time $t_w \in [t^*, t^* + 2d)$. Thus, node each $w \in G$ transitions from RECOVER to PULSE at time t_w, when Guard G2 becomes satisfied. Finally, Lemma 7 also guarantees that $t^* < t_0 + T_{\text{consensus}}/\vartheta \leq T_{\text{active}} + \rho + T_{\text{consensus}}/\vartheta$. □

We will now establish a series of lemmas in order to show that we can either apply Lemma 8 directly or in conjunction with Lemma 9 to guarantee stabilisation. In the remainder of this section, we define the following abbreviations:

$$\begin{aligned}
\alpha &= T_{\text{listen}} + d \\
\beta &= \alpha + T_{\text{wait}} + \gamma + 4T_1 + 3d + 2\delta \\
\beta' &= (T_{\text{active}} - T_2 - T_{\text{consensus}})/\vartheta \\
\delta &= \max\{T_1 + T_{\text{wait}}, T_1 + d + T_{\text{listen}} + T_2 + T_{\text{consensus}}\} \\
\gamma &= T_2/\vartheta - 5T_1 - 3d.
\end{aligned}$$

Remark 5. We have that $\gamma < \delta < \beta \leq \beta' < T_{\text{active}}/\vartheta$, where the inequality $\beta \leq \beta'$ is equivalent to Constraint (14).

In the following, we consider the time intervals $[\alpha, \beta]$ and $[\alpha, \beta']$. We distinguish between two cases and show that in either case stabilisation is ensured. The cases are:

- no correct node transitions to INPUT 1 during $[\alpha, \beta]$, and
- some correct node transitions to INPUT 1 during $[\alpha, \beta']$.

We start with the latter case.

Lemma 10. *Suppose node $v \in G$ transitions to* INPUT 1 *at time $t \in [\alpha, \beta')$. Then there exists a time $t' \in [t, T_{active}/\vartheta - 2d)$ such that one of the following holds:*

1. *every $u \in G$ satisfies $p(u, t_u) = 1$ for some $t_u \in [t', t' + 2d)$, or*
2. *every $u \in G$ is in state* RECOVER *and* LISTEN *at time $t' + 2d$ and Guard G4 is not satisfied at u during $[t' + 2d, t^* + 2d]$, where $t^* := \min_{w \in G}\{p(w, t' + 2d)\}$.*

Proof. By Constraint (6), we have $T_1/\vartheta > 2d$. As $\alpha \leq t < \beta' < (T_{\text{active}} - T_2)/\vartheta$, we can apply Lemma 6 to time t. Due to Constraint (8), we can apply Lemma 7 with time $t_0 = t - T_{\text{listen}} - d + T_2/\vartheta$, yielding a time $t' < \beta' - T_{\text{listen}} - 3d + T_2/\vartheta + T_{\text{consensus}}/\vartheta < T_{\text{active}}/\vartheta - 2d$ such that each $u \in G$ transitions to the same output state OUTPUT 0 or OUTPUT 1 in the auxiliary state machine at time $t_u \in [t', t' + 2d)$. If this state is OUTPUT 1, each $u \in G$ transitions to OUTPUT 1 at time $t_u \in [t', t' + 2d)$. This implies that Guard G2 is activated and u transitions to PULSE so that $p(u, t_u) = 1$. If this state is OUTPUT 0, Guard G2' implies that each $u \in G$ either remains in or transitions to state RECOVER at time t_u. Moreover, node u immediately transitions to state LISTEN in the auxiliary state machine. Finally, note that Lemma 6 also states that Guard G4 cannot be satisfied again before time $t^* + T_1/\vartheta > t^* + 2d$. □

Corollary 4. *Suppose node $v \in G$ transitions to* INPUT 1 *at time $t \in [\alpha, \beta']$. Then there exists $t' < T_{active} + \rho + T_{consensus}/\vartheta$ such that every $u \in G$ satisfies $p(v, t_u) = 1$ for some $t_u \in [t', t' + 2d)$.*

Proof. We apply Lemma 10. In the first case of Lemma 10, the claim immediately follows. In the second case, it follows by applying Lemma 9. □

We now turn our attention to the other case, where no node $v \in G$ transitions to INPUT 1 during the time interval $[\alpha, \beta]$.

Lemma 11. *If no $v \in G$ transitions to* INPUT 1 *during $[\alpha, \beta]$, then no $v \in G$ transitions to state* EXEC 1 *during $[\alpha + T_1 + T_{wait}, \beta)$.*

Proof. Observe that any $v \in G$ that does not transition to PULSE during $[\alpha, \alpha + T_1 + T_{\text{wait}}]$ must transition to RECOVER at some time from that interval once Guard G2' is satisfied. Note that leaving state RECOVER requires Guard G2 to be satisfied, and hence, a transition to OUTPUT 1 in the auxiliary state machine. However, as no node $v \in G$ transitions to INPUT 1 during $[\alpha, \beta)$, it follows that each $v \in G$ that is in state INPUT 1 in the auxiliary state machine at time $t \in [\alpha + T_1 + T_{\text{wait}}, \beta)$ is also in state RECOVER in the main state machine. Thus, Guard G6 cannot be satisfied. We conclude that no correct node transitions to state EXEC 1 during the interval $[\alpha + T_1 + T_{\text{wait}}, \beta)$. □

We now show that if Guard G4 cannot be satisfied for some time, then there exists an interval during which no correct node is participating in or attempting to simulate a consensus instance.

Lemma 12. *Let $t \in (d, \beta - 2\delta)$ and suppose Guard G4 is not satisfied during the interval $[t, t + \gamma]$. Then there exists a time $t' \in [t, \beta - \delta]$ such that no $v \in G$ is in state* EXEC 0 *or* EXEC 1 *during $(t' - d, t']$.*

Proof. Observe that Guard G3 cannot be satisfied before time $t + \gamma + T_2/\vartheta < T_{\text{active}}/\vartheta$ at any correct node, and by the assumption, Guard G4 is not satisfied during the interval $[t, t + \gamma]$. We proceed via a case analysis.

First, suppose $v \in G$ is in state LISTEN at time t. As Guard G3 or Guard G4 cannot be satisfied during $[t, t + \gamma]$ node v remains in LISTEN until time $t + \gamma$. Moreover, if v leaves LISTEN during $[t + \gamma, t + \gamma + T_2/\vartheta]$, it must do so by transitioning to READ. Hence, it cannot transition to EXEC 0 or EXEC 1 before Guard G6, Guard G6' or Guard G7 are satisfied. Either way, v cannot reach states EXEC 0 or EXEC 1 before time $t + \gamma + T_2/\vartheta$. Hence, in this case, v is not in the two execution states during $I_1 = [t, t + \gamma + T_2/\vartheta) = [t, t + 2T_2/\vartheta - 3(T_1 + d))$.

Let us consider the second case, where v is not in LISTEN at time t. Note that the timer T_2 cannot be reset at v during the interval $[t + T_{\text{listen}}, t + \gamma]$, as this can happen only if v transitions to INPUT 0 or INPUT 1 and Guard G4 cannot be satisfied during $[t, t + \gamma]$. Hence, the only way for this to happen is if v transitions from READ to either INPUT 0 or INPUT 1 during the interval $[t, t + T_{\text{listen}}]$.

It follows that v cannot *transition* to EXEC 0 or EXEC 1 during the interval $(t + T_{\text{listen}} + T_2, t + \gamma + T_2/\vartheta)$. Moreover, if v transitions to these states before $t + T_{\text{listen}} + T_2$, then v must transition away from them by time $t + T_{\text{listen}} + T_2 + T_{\text{consensus}}$, as Guard G8 or Guard G9 become satisfied. Therefore, node v is in LISTEN, READ, INPUT 0 or INPUT 1 during $I_2 = [t + T_{\text{listen}} + T_2 + T_{\text{consensus}}, t + \gamma + T_2/\vartheta)$. Applying Constraint (9), we get that

$$T_{\text{listen}} + T_2 + T_{\text{consensus}} < 2T_2/\vartheta - 5T_1 - 4d,$$

and hence, by setting $t' = t + T_{\text{listen}} + T_2 + T_{\text{consensus}} + d < \beta - \delta$, we get that $(t' - d, t'] \subseteq I_1 \cap I_2$. That is, in either case v is not in state EXEC 0 or EXEC 1 during this short interval. □

Lemma 13. *There exists $t \in [\alpha + T_1 + T_{wait} + d, \alpha + 4T_1 + T_{wait} + 3d + \gamma) \subset (d, \beta - 2\delta)$ such that Guard G4 is not satisfied during $[t, t + \gamma]$ at any $v \in G$.*

Proof. Let $t' \in [\alpha + T_1 + T_{\text{wait}}, \alpha + T_1 + T_{\text{wait}} + d + \gamma]$ be a time when some $v \in G$ transitions to state WAIT. If such t' does not exist, the claim trivially holds for $t = \alpha + T_1 + T_{\text{wait}} + d$. Otherwise, since $t' \leq T_{\text{listen}} + T_1 + T_{\text{wait}} + 2d + \gamma \leq (T_{\text{active}} - T_2)/\vartheta$ by Constraint (13), we can apply Lemma 5 to time t', which yields that no $u \in G$ transitions to WAIT during

$$[t' + 3T_1 + d, t' + T_2/\vartheta - 2T_1 - d) = [t - d, t + \gamma),$$

where $t = t' + 3T_1 + 2d$. Thus, Guard G4 is not satisfied at any $v \in G$ during $[t, t + \gamma]$. □

Lemma 14. *Suppose no $v \in G$ transitions to* INPUT 1 *during $[\alpha, \beta]$. Then no $v \in G$ transitions to* PULSE *during $[\beta - \delta, \beta]$.*

Proof. First, we apply Lemma 13 to obtain an interval $[t, t+\gamma] \subseteq [\alpha + T_1 + T_{\text{wait}} + d, \beta - 2\delta]$ during which Guard G4 cannot be satisfied at any $v \in G$. Applying Lemma 12 to this interval yields an interval $(t' - d, t'] \subset (t, \beta - \delta]$ during which no $v \in G$ is in state EXEC 0 or EXEC 1. This implies that no node is running a consensus instance during this time interval, and moreover, no messages from prior consensus instances are in transit to or arrive at any correct node at time t'. In particular, any $v \in G$ that (attempts to) simulate a consensus instance at time t' or later must first reinitialise the simulation by transitioning to EXEC 0 or EXEC 1 first.

Next, let us apply Lemma 11, to see that no v transitions to EXEC 1 during the interval $[\alpha + T_1 + T_{\text{wait}}, \beta] \supset [t', \beta]$. Thus, if any node $v \in G$ attempts to simulate $\mathbf{C}$, it must start the simulation by transitioning to EXEC 0. This entails that any correct node attempting to simulate $\mathbf{C}$ does so with input 0. Because $\mathbf{C}$ is a silent consensus routine (see Definition 1), this entails that v does not send any message related to $\mathbf{C}$ unless it receives one from a correct node first, and in absence of such a message, it will not terminate with output 1. We conclude that no $v \in G$ transitions to OUTPUT 1 during $[t', \beta] \subseteq [\beta - \delta, \beta]$. The claim follows by observing that a transition to PULSE in the main state machine requires a transition to OUTPUT 1 in the auxiliary state machine. □

Lemma 15. *Suppose no $v \in G$ transitions to* PULSE *during $[\beta - \delta, \beta]$. Then at time β every $v \in G$ is in state* RECOVER *in the main state machine and state* LISTEN *in the auxiliary state machine. Moreover, Guard G4 is not satisfied during $[\beta, t^* + 2d)$, where $t^* = \max_{v \in G}\{p_{next}(v, \beta)\}$.*

Proof. As no $v \in G$ transitions PULSE during $[\beta - \delta, \beta]$, either Guard G1 or Guard G2' lead each $v \in G$ to RECOVER. More precisely, every $v \in G$ is in state RECOVER of the main state machine during $[\beta - \delta + T_1 + T_{\text{wait}}, \beta]$. Next, observe that Guard G4 is not satisfied during $[\beta - \delta + T_1 + d, \beta]$, implying that each $v \in G$ is in state LISTEN of the auxiliary state machine during $[\beta - \delta + T_1 + d + T_{\text{listen}} + T_2 + T_{\text{consensus}}, \beta]$. Recalling that

$$\delta = \max\{T_1 + T_{\text{wait}}, T_1 + d + T_{\text{listen}} + T_2 + T_{\text{consensus}}\},$$

the first claim of the lemma follows. For the second claim, observe that Guard G4 cannot be satisfied before the next transition to WAIT by a correct node occurs. This cannot happen before T_1/ϑ time has passed after a correct node transitioned to PULSE by Guard G1. Since $T_1/\vartheta > 2d$ by Constraint (6), the claim follows. □

We can now show that if no INPUT 1 transitions occur during $[\alpha, \beta]$, then all nodes end up in the RECOVER state in the main state machine before T_{active} timeout expires at any node. This will eventually activate Guard G3 at every correct node, leading to a correct simulation of $\mathbf{C}$ with all 1 inputs.

Corollary 5. *Suppose no $v \in G$ transitions to* INPUT 1 *during $[\alpha, \beta]$. Then there exists a time $t < T_{active} + \rho + T_{consensus}$ such that every $v \in G$ transitions to* PULSE *at time $t_v \in [t, t + 2d)$.*

Proof. By Lemma 14 and Lemma 15, the prerequisites of Lemma 9 are satisfied at time $t = \beta < T_{\text{active}}/\vartheta$. □

It remains to show that the constraints in Table 3 can be satisfied for some $\vartheta > 1$ such that all timeouts are in $O(R)$.

Lemma 16. *Let $1 < \vartheta < (1 + \sqrt{5})/3 \approx 1.0787$. The constraints in Table 3 can be satisfied. Moreover, all timeouts are $O(R)$.*

Proof. Recall that R is the number of rounds the consensus routine needs to declare output and $T(R)$ is time required to simulate the consensus routine. We parametrize T_2 and T_{active} using X

and Y, where we require that $X \in \Theta(R)$ is chosen so that $T(R)/X \leq \varepsilon$, for a constant $\varepsilon > 0$ to be determined later. We then can set

$$\begin{aligned}
T_1 &:= 3\vartheta d\\
T_{\text{listen}} &:= 3\vartheta^2 d\\
T_2 &:= X\\
\tau &:= \max\left\{(1-1/\vartheta)X + (3\vartheta^2+9\vartheta+3)d, (1-1/\vartheta)Y+\rho\right\}\\
T_{\text{consensus}} &:= \vartheta(\tau+\varepsilon X)\\
T_{\text{wait}} &:= X + T_{\text{consensus}}\\
T_{\text{active}} &:= Y,
\end{aligned}$$

immediately satisfying Constraint (6), Constraint (7), Constraint (10), Constraint (11), and Constraint (12). Moreover, Constraint (8) holds by requiring that X is at least a sufficiently large constant.

For the remaining constraints, denote by $C(\vartheta, d)$ a sufficiently large constant subsuming all terms that depend on ϑ and d only and abbreviate $T'_{\text{consensus}} := (\vartheta-1)\max\{X,Y\} + \varepsilon\vartheta X$ (i.e., the non-constant terms of $T_{\text{consensus}}$). To satisfy Constraint (9), Constraint (13), and Constraint (14), it is then sufficient to guarantee that

$$\begin{aligned}
(2/\vartheta-1)X &> T'_{\text{consensus}} + C(\vartheta,d)\\
Y &\geq 5X + T'_{\text{consensus}} + C(\vartheta,d)\\
Y &\geq 2X + T'_{\text{consensus}} + \vartheta(4X + 4T'_{\text{consensus}}) + C(\vartheta,d),
\end{aligned}$$

where the second inequality automatically holds if the third is satisfied. We also note that $Y \geq X$ is a necessary condition to satisfy the third inequality, implying that we may assume that $T'_{\text{consensus}} = (\vartheta-1)Y + \varepsilon X$. We rearrange the remaining inequalities, yielding

$$\begin{aligned}
(2/\vartheta-1-\varepsilon)X &> (\vartheta-1)Y + C(\vartheta,d)\\
(2+3\vartheta-4\vartheta^2)Y &\geq (2+4(1+\varepsilon)\vartheta)X + C(\vartheta,d). \qquad (1)
\end{aligned}$$

Recall that ϑ and $C(\vartheta,d)$ are constant, and ε is a constant under our control. Hence, these inequalities can be satisfied if and only if

$$(2/\vartheta-1)(2+3\vartheta-4\vartheta^2) > (2+4\vartheta)(\vartheta-1).$$

The above inequality holds for all $\vartheta \in (1,(1+\sqrt{5})/3)$. Note that, as $\vartheta, \varepsilon, C(\vartheta,d) \in O(1)$, we can choose $X \in \Theta(R)$ as initially stated, implying that all timeouts are in $O(R)$, as desired. □

Corollary 6. *For $\vartheta < (1+\sqrt{5})/3$ and suitably chosen timeouts, in any execution there exists $t \in O(R)$ such that every $v \in G$ transitions to* PULSE *during the time interval $[t, t+2d)$.*

Proof. We choose the timeouts in accordance with Lemma 16. If no $v \in G$ transitions to INPUT 1 during $[\alpha,\beta]$, Corollary 5 yields the claim. Otherwise, some node $v \in G$ transitions to state INPUT 1 during the interval $[\alpha,\beta] \subseteq [\alpha,\beta']$ and the claim holds by Corollary 4. □

8.6 Proof of Theorem 2

We observe that the accuracy bounds can be set to be within a constant factor apart from each other.

Corollary 7. *Let $\vartheta < (1+\sqrt{5})/3$ and $\varphi_0(\vartheta) = 1+6(\vartheta-1)/(2+3\vartheta-4\vartheta^2) \in 1+O(\vartheta-1)$. For any constant $\varphi > \varphi_0(\vartheta)$, we can obtain accuracy bounds satisfying $\Phi^+/\Phi^- \leq \varphi$ and $\Phi^-, \Phi+ \in \Theta(R)$.*

Proof. By Lemma 8, the accuracy bounds we get from the constuction are $\Phi^- = T_2/\vartheta$ and $\Phi^+ = (T_2 + T_{\text{consensus}})/\vartheta$. Choosing the timeouts as in the proof of Lemma 16, we get that $\Phi^+/\Phi^- = 1 + (\vartheta - 1)Y/X + \varepsilon$, where ε is an arbitrarily small constant. Checking Inequality (1), we see that we can choose $Y/X = (2 + 4(1 + \varepsilon))/(2 + 3\vartheta - 4\vartheta^2)$. Choosing ε sufficiently small, the claim follows. □

Now we are ready to prove our main theorem of this section.

Theorem 2. *Let $f \geq 0$, $n > 3f$ and $(1 + \sqrt{5})/3 > \vartheta > 1$. Suppose for a network of n nodes there exist*

- *an f-resilient synchronous consensus algorithm $\mathbf{C}$, and*
- *an f-resilient resynchronisation algorithm $\mathbf{B}$ with skew $\rho \in O(d)$ and sufficiently large separation window $\Psi \in O(R)$ that tolerates clock drift of ϑ,*

where $\mathbf{C}$ runs in $R = R(f)$ rounds and lets nodes send at most $M = M(f)$ bits per round. Then a $\varphi_0(\vartheta) \in 1 + O(\vartheta - 1)$ exists so that for any constant $\varphi > \varphi_0(\vartheta)$ and sufficiently large $T \in O(R)$, there exists an f-resilient pulse synchronisation algorithm $\mathbf{A}$ for n nodes that

- *has skew $\sigma = 2d$ and satisfies the accuracy bounds $\Phi^- = T$ and $\Phi^+ = T\varphi$,*
- *stabilises in $T(\mathbf{B}) + O(R)$ time and has nodes broadcast $M(\mathbf{B}) + O(M)$ bits per time unit.*

Proof. By the properties of the resynchronisation algorithm $\mathbf{B}$, we get that the a good resynchronisation pulse occurs within time $T(\mathbf{B})$. Once this happens, Corollary 6 shows all correct nodes transition to PULSE during $[t, t + 2d)$ for $t \in T(\mathbf{B}) + O(R)$. By Lemma 8 we get that the algorithm stabilises by time t and has skew $\sigma = 2d$. From Corollary 7 we get that the accuracy bounds can be set to be within factor φ apart without affecting the stabilisation time asymptotically.

To analyse the number of bits sent per time unit, first observe that the main state machine communicates whether a node transitions to PULSE or WAIT. This can be encoded using messages of size $O(1)$. Moreover, as node remains $\Omega(d)$ time in PULSE or WAIT, the main state machine sends only $O(1)$ bits per time unit. Second, the auxiliary state machine does not communicate apart from messages related to the simulation of consensus. The non-self-stabilising pulse synchronisation algortihm sends messages only when a node generates a pulse and the time between pulses is $\Omega(d)$. Thus, while simulating $\mathbf{C}$, each node broadcasts at most $M(\mathbf{C}) + O(1)$ bits per time unit. □

9 Resynchronisation algorithms

In this section, we give the second key component in the proof of Theorem 1. We show that given *pulse synchronisation* algorithms for networks of small size and with low resilience, it is possible to obtain *resynchronisation* algorithms for large networks and with high resilience. More precisely, we show the following theorem.

Theorem 3. *Let $f, n_0, n_1 \in \mathbb{N}$, $n = n_0 + n_1$, $f_0 = \lfloor (f-1)/2 \rfloor$, $f_1 = \lceil (f-1)/2 \rceil$, and $1 < \vartheta \leq 1.007$. Suppose that for some given $\Psi \in \Omega(1)$, sufficiently small constant $\varphi > \varphi_0(\vartheta)$, and $T_0 \in \Theta(\Psi)$, it holds that for any $h \in \{0, 1\}$ and $T_0 \leq T \in O(\Psi)$ there exists a pulse synchronisation algorithm $\mathbf{A}_h$ that*

- *runs on n_h nodes and tolerates f_h faulty nodes,*
- *has skew $\sigma = 2d$ and accuracy bounds $\Phi_h^- = T$ and $\Phi_h^+ = T\varphi$.*

Then there exists a resynchronisation algorithm with skew $\rho \in O(d)$ and separation window of length Ψ that generates a resynchronisation pulse by time $\max\{T(\mathbf{A}_0), T(\mathbf{A}_1)\} + O(\Psi)$*, where nodes broadcast only $O(1)$ additional bits per time unit.*

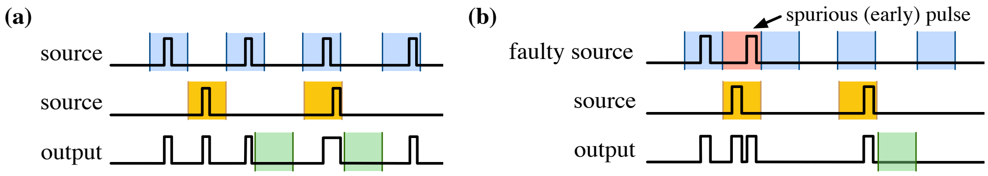


Figure 9: Idea of the resynchronisation algorithm. We take two pulse sources with coprime frequencies and output the logical OR of the two sources. In this example, the pulses of the first source should occur in the blue regions, whereas the pulses of the second source should hit the yellow regions. The green regions indicate a period where a pulse from either source is followed by at least Ψ time of silence. Eventually, such a region appears. (a) Two correct sources that pulse with set frequencies. (b) One faulty source that produces spurious pulses. Here, a pulse occurs too early (red region), and thus, we then enforce that the faulty source is silenced for $\Theta(\Psi)$ time. Once the faulty source is silenced, the correctly working source has time to produce a good resynchronisation pulse, that is, a pulse in the output signal that is followed by at least Ψ time of silence.

9.1 The high-level idea

Our goal is to devise a self-stabilising resynchronisation algorithm with skew $\rho \in O(d)$ and separation window Ψ for n nodes that tolerates $f < n/3$ faulty nodes. That is, we want an algorithm that guarantees that there exists a time t such that all correct nodes locally generate a single resynchronisation pulse during the interval $[t, t+\rho)$ and no new pulse during the interval $[t+\rho, t+\rho+\Psi)$. Note that a correct resynchronisation algorithm is also allowed to generate various kinds of spurious resynchronisation pulses, such as pulses that are followed by a new resynchronisation pulse too soon (i.e., before Ψ time units have passed) or pulses that are only generated by a subset of the correct nodes.

The algorithm idea. In order to illustrate the idea behind our resynchronisation algorithm, let us ignore clock drift and suppose we have two sources of pulses that generate pulses with fixed frequencies. Whenever either source generates a pulse, then a resynchronisation pulse is triggered as well. If the sources generate pulses with frequencies that are coprime multiples of (a sufficiently large) $C \in \Theta(\Psi)$, then we are guaranteed that eventually one of the sources produces a pulse followed by at least Ψ time units before a new pulse is generated by either of the two sources. See Figure 9 for an illustration.

Put otherwise, suppose $p_h(v,t) \in \{0,1\}$ indicates whether node v observe the pulse source $h \in \{0,1\}$ generating a pulse at time t. Using the above scheme, the output variable for the resynchronisation algorithm would be $r(v,t) = \max\{p_h(v,t)\}$. If, eventually, each source h generates a pulse roughly every C_h time units, setting $C_0 < C_1$ to be coprime integer multiples of $C \in \Theta(\Psi)$ (we allow for a constant-factor slack to deal with clock drift, etc.), we eventually have a time when a pulse is generated by one source, but not source will generate another pulse for at least Ψ time units.

Obviously, if we had such reliable self-stabilising pulse sources for n nodes and $f < n/3$ faulty nodes, then we would have effectively solved the pulse synchronisation problem already. However, our construction given in Section 8 relies on having a resynchronisation algorithm. Thus, in order to avoid this chicken-and-egg problem, we partition the set of n nodes into two and have each part run an instance of a pulse synchronisation algorithm with resilience almost $f/2$. That is, we take two *pulse synchronisation algorithms* with low resilience, and use these to obtain a *resynchronisation algorithm* with high resilience. This allows us to recursively construct resynchronisation algorithms starting from trivial pulse synchronisation algorithms that do not tolerate any faulty nodes.

The final obstacle to this construction is that we cannot guarantee that *both* instances with smaller resilience operate correctly, as the total number of faults exceeds the number that can be tolerated by each individual instance. We overcome this by enlisting the help of *all* nodes to check, for each instance, whether its output appears to satisfy the desired frequency bounds. If not, its output is conservatively filtered out (for a sufficiently large period of time) by a voting mechanism. This happens only for an incorrect output, implying that the fault threshold for the respective instance must have been exceeded. Accordingly, the other instance is operating correctly and, thanks to the absence of further interference from the faulty instance, succeeds in generating a resynchronisation pulse.

Using two pulse synchronisers. We now overview how to use two pulse synchronisation algorithms to implement our simple resynchronisation algorithm described above. Let

$$n_0 = \lfloor n/2 \rfloor \text{ and } n_1 = \lceil n/2 \rceil$$
$$f_0 = \lfloor (f-1)/2 \rfloor \text{ and } f_1 = \lceil (f-1)/2 \rceil.$$

Observe that we have $n = n_0 + n_1$ and $f = f_0 + f_1 + 1$. We partition the set V of n nodes into two sets V_h for $h \in \{0,1\}$ such that $V = V_0 \cup V_1$, where $V_0 \cap V_1 = \emptyset$ and $|V_h| = n_h$. We now pick two pulse synchronisation algorithms $\mathbf{A}_0$ and $\mathbf{A}_1$ with the following properties:

- $\mathbf{A}_h$ runs on n_h nodes and tolerates f_h faulty nodes,
- $\mathbf{A}_h$ stabilises in time $T(\mathbf{A}_h)$ and has nodes send $M(\mathbf{A}_h)$ bits per time unit, and
- $\mathbf{A}_h$ has skew $\sigma \in O(d)$ and accuracy bounds $\Phi_h = (\Phi_h^-, \Phi_h^+)$, where $\Phi_h^-, \Phi_h^+ \in O(\Psi)$.

We let the nodes in set V_h execute the pulse synchronisation algorithm $\mathbf{A}_h$.

An optimistic approach would be to use each $\mathbf{A}_h$ as a source of pulses by checking whether at least $n_h - f_h$ nodes in the set V_h generated a pulse within a time window of (roughly) length $\sigma = 2d$. Unfortunately, we cannot directly use the pulse synchronisation algorithms $\mathbf{A}_0$ and $\mathbf{A}_1$ as reliable sources of pulses. There can be a total of $f = f_0 + f_1 + 1 < n/3$ faulty nodes, and thus, it may be that for one $h \in \{0,1\}$ the set V_h contains more than f_h faulty nodes. Hence, the algorithm $\mathbf{A}_h$ may never stabilise and can generate spurious pulses at uncontrolled frequencies. In particular, the algorithm may always generate pulses with frequency less than Ψ preventing our simple solution from working. However, we are guaranteed that at least one of the algorithms stabilises.

Lemma 17. *If there are at most f faulty nodes, then there exists $h \in \{0,1\}$ such that $\mathbf{A}_h$ stabilises by time $T(\mathbf{A}_h)$.*

Proof. Observe that $f_0 + f_1 + 1 = f$. In order to prevent both algorithms from stabilising, we need to have at least $f_0 + 1$ faults in the set V_0 and $f_1 + 1$ faults in the set V_1, totalling $f_0 + f_1 + 2 > f$ faults in the system. □

Once the algorithm $\mathbf{A}_h$ for some $h \in \{0,1\}$ stabilises, we have at least $n_h - f_h$ correct nodes in set V_h locally generate pulses with skew σ and accuracy bounds $\Phi_h = (\Phi_h^-, \Phi_h^+)$. However, it may be that the other algorithm $\mathbf{A}_{1-h}$ never stabilises. Moreover, the algorithm $\mathbf{A}_{1-h}$ may forever generate spurious pulses at arbitrary frequencies. Here, a spurious pulse refers to any pulse that does not satisfy the skew σ and accuracy bounds of $\mathbf{A}_{1-h}$. For example, a spurious pulse may be a pulse that only a subset of nodes generate, one with too large skew, or a pulse that occurs too soon or too late.

In order to tackle this problem, we employ a series of threshold votes and timeouts to filter out any spurious pulses generated by an unstabilised algorithm *that violate timing constraints.* This way, we can impose some control on the frequency at which an unstabilised algorithm

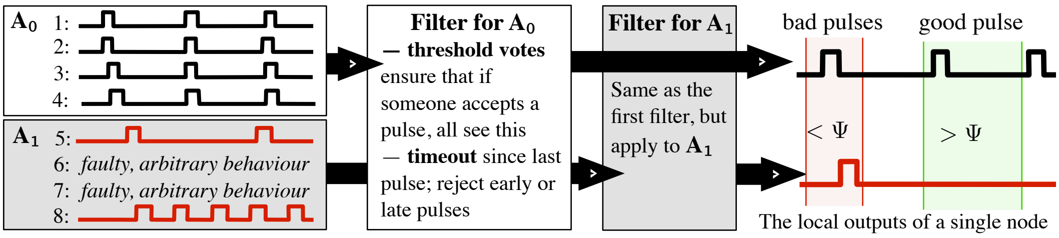

A₀
1:
2:
3:
4:
A₁
5:
6: faulty, arbitrary behaviour
7: faulty, arbitrary behaviour
8:
Filter for A₀
— threshold votes ensure that if someone accepts a pulse, all see this
— timeout since last pulse; reject early or late pulses
Filter for A₁
Same as the first filter, but apply to A₁
bad pulses
good pulse
< Ψ
> Ψ
The local outputs of a single node

Figure 10: Example of the resynchronisation construction for 8 nodes tolerating 2 faults. We partition the network into two parts, each running a pulse synchronisation algorithm $\mathbf{A}_i$. The output of $\mathbf{A}_i$ is fed into the respective filter and any pulse that passes the filtering is used as a resynchronisation pulse. The filtering consists of (1) having *all* nodes in the network participate in a threshold vote to see if anyone thinks a pulse from $\mathbf{A}_i$ occurred (i.e. enough nodes running $\mathbf{A}_i$ generated a pulse) and (2) keeping track when was the last time a pulse from $\mathbf{A}_i$ occurred to check that the accuracy bounds of $\mathbf{A}_i$are respected: pulses that appear too early or too late are ignored. Moreover, if $\mathbf{A}_i$ generates pulses at incorrect frequencies, the filtering mechanism blocks all pulses generated by $\mathbf{A}_i$ for $\Theta(\Psi)$ time.

may trigger resynchronisation pulses. As long as these frequency bounds are satisfied, it is inconsequential if a non-stabilised algorithm triggers resynchronisation pulses at a subset of the nodes only. We want our filtering scheme to eventually satisfy the following properties:

- If $\mathbf{A}_h$ has stabilised, then all pulses generated by $\mathbf{A}_h$ are accepted.
- If $\mathbf{A}_h$ has not stabilised, then only pulses that respect given frequency bounds are accepted.

More precisely, in the first case the filtered pulses respect (slightly relaxed) accuracy bounds Φ_h of $\mathbf{A}_h$. In the second case, we enforce that the filtered pulses must either satisfy roughly at the same accuracy bounds Φ_h or they must be sufficiently far apart. That is, the nodes will reject any pulses generated by $\mathbf{A}_h$ if they occur either too soon or too late.

Once we have the filtering mechanism in place, it becomes relatively easy to implement our conceptual idea for the resynchronisation algorithm. We apply the filtering mechanism for both algorithms $\mathbf{A}_0$ and $\mathbf{A}_1$ and use the filtered outputs as a source of pulses in our algorithm, as illustrated in Figure 10. We are guaranteed that at least one of the sources eventually produces pulses with well-defined accuracy bounds. Furthermore, we know that also the other source must either respect the given accuracy bounds or refrain from generating pulses for a long time. In the case that both sources respect the accuracy bounds, we use the coprimality of frequencies to guarantee a sufficiently large separation window for the resynchronisation pulses. Otherwise, we exploit the fact that the unreliable source stays silent for sufficiently long for the reliable source to generate a pulse with a sufficiently large separation window.

9.2 Filtering spurious pulses

Our pulse filtering scheme follows a similar idea as our recent construction of synchronous counting algorithms [26]. However, considerable care is needed to translate the approach from the (much simpler) synchronous round-based model to the bounded-delay model with clock drift. We start by describing the high-level idea of the approach before showing how to implement the filtering scheme in the bounded-delay model considered in this work.

For convenience, we refer to the nodes in set V_h as *block h*. First, for each block $h \in \{0, 1\}$ every node $v \in G$ performs the following threshold vote:

1. If v observes at least $n_h - f_h$ nodes in V_h generating a pulse, vote for generating a resynchronisation pulse.

2. If at least $n - f$ nodes in V voted for pulse by block h (within the time period this should take), then v accepts it.

The idea here is that if some correct node *accepts* a pulse in Step 2, then every correct node must have seen at least $n - 2f \geq f + 1$ *votes* due to Step 1. Moreover, once a node observes at least $f + 1$ votes, it can deduce that some correct node saw at least $n_h - f_h$ nodes in block h generate a pulse. Thus, if any correct node accepts a pulse generated by block h, then all correct nodes are aware that a pulse *may* have happened.

Second, we have the nodes perform temporal filtering by keeping track of when block h last (may have) generated a pulse. To this end, each node has a local "cooldown timer" that is reset if the node suspects that block h has not yet stabilised. If a pulse is accepted by the above voting mechanism, then a resynchronisation pulse is triggered if the following conditions are met:

1. the cooldown timer has expired, and
2. sufficiently long time has passed since the most recent pulse from h.

A correct node $v \in G$ resets its cooldown timer if it

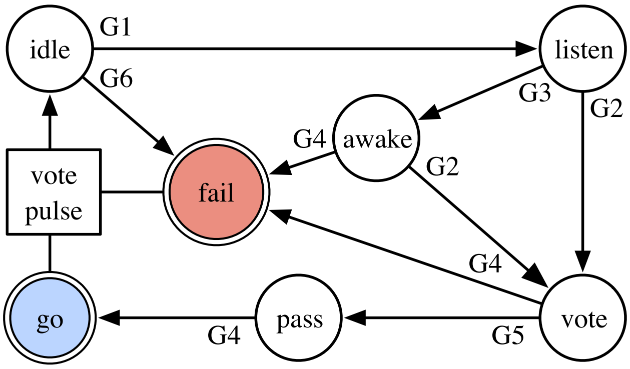
1. observes at least $f + 1$ votes for a pulse from block h, but not enough time has passed since v last saw at least $f + 1$ votes,
2. observes at least $f + 1$ votes, but not $n - f$ votes in a timely fashion, or
3. has not observed a pulse from block h for too long, that is, block h should have generated a new pulse by now.

Thus, whenever a block $h \in \{0, 1\}$ triggers a resynchronisation pulse at node $v \in G$, then each node $u \in G$ either resets its cooldown timer or also triggers a resynchronisation pulse. Furthermore, if $v \in G$ does not observe a pulse from block h within the right time window, it will also reset its cooldown counter. Finally, each node refuses to trigger a resynchronisation pulse when its cooldown timer is active. Note that if $\mathbf{A}_h$ stabilises, then eventually the cooldown timer for block h expires and is not reset again. This ensures that eventually at least one of the blocks triggers resynchronisation pulses.

Implementation in the bounded-delay model. We implement the threshold voting and temporal filtering with two state machines depicted in Figure 11 and Figure 12. For each block $h \in \{0, 1\}$, every node runs a single copy of the voter and validator state machines in parallel (i.e., nodes essentially run a product of the two state machines). In the voter state machine given in Figure 11, there are two key states, FAIL and GO, which are used to indicate a local signal for the validator state machine in Figure 12.

The key feature of the voter state machine is the voting scheme: if some node $v \in G$ transitions from VOTE to GO, then all nodes must transition to either FAIL or GO. This is guaranteed by the fact that a node only transitions to GO if it has observed at least $n - f$ nodes in the state VOTE within a short time window. This in turn implies that all nodes must observe at least $n - 2f > f$ nodes in VOTE (in a slightly larger time window). Thus, any node in state IDLE must either transition directly to FAIL from IDLE or move on to LISTEN. If a node transitions to state LISTEN, then it is bound to either transition to GO or FAIL.

The validator state machine in turn ensures that any subsequent GO transitions of v are at least $T_{\min,h}/\vartheta$ or $T_{\text{cool}}/\vartheta$ time units apart. Moreover, if any FAIL transition occurs at time t, then any subsequent GO transition can occur at time $t + T_{\text{cool}}/\vartheta$ at earliest. The voter state machine also handles generating a FAIL transition if the underlying pulse synchronisation algorithm does not produce a pulse within time $T_{\max,h}$. This essentially forces the TRIGGER transitions in the validator state machine to occur between accuracy bounds $\Lambda_h^- \approx T_{\min,h}/\vartheta$ and $\Lambda_h^+ \approx T_{\max,h}$ or at least $T_{\text{cool},h}/\vartheta$ time apart. Furthermore, if the underlying pulse synchronisation algorithm $\mathbf{A}_h$ stabilises, then the TRIGGER transitions roughly follow the accuracy bounds of $\mathbf{A}_h$.



Transition guards:

A $\equiv$ $\geq n_h - f_h$ pulse messages from V_h

B $\equiv$ $> f$ vote messages from V within T_{att}

G1 $\equiv$ A or B

G2 $\equiv$ A

G3 $\equiv$ $\neg A$ and B

G4 $\equiv$ $\langle T_{\mathrm{vote}}, \text{WAKE-UP} \rangle$ expires

G5 $\equiv$ $\geq n - f$ vote messages within time T_{vote}

G6 $\equiv$ $\langle T_{\mathrm{max,h}} \rangle$ expires

Figure 11: The voter state machine is the first of the two state machines used to trigger resynchronisation pulses by block $h \in \{0, 1\}$. Every node runs a separate copy of the voter machine for both blocks. The voter state machine performs a threshold vote to ensure that if at *some* node a resynchronisation pulse is (or might be) triggered by block h, then *all* correct nodes see this by observing at least $f + 1$ VOTE messages within T_{att} local time. The box before IDLE indicates that the VOTE and pulse flags are cleared whenever a node enters the state IDLE. In particular, this also explicitly clears the buffers used in the transition guards. Note that the FAIL and GO transitions are immediate, as there are no transition guards away from these states. The two states are used to signal the validator state machine given in Figure 12 to generate resynchronisation pulses or to refrain from doing so until the cooldown timer expires.

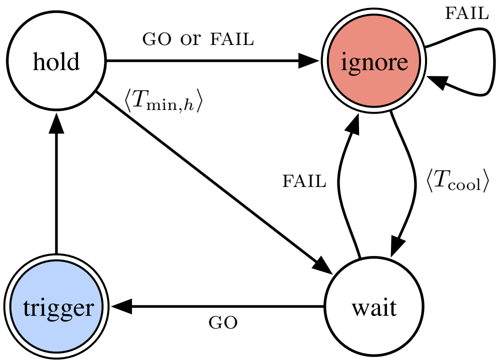

hold
GO or FAIL
ignore
FAIL
$\langle T_{\mathrm{min},h} \rangle$
FAIL
$\langle T_{\mathrm{cool}} \rangle$
trigger
GO
wait

Figure 12: The validator state machine is the second of the two state machines used to trigger resynchronisation pulses by block $h \in \{0, 1\}$. Every node runs a separate copy of the validator state machine for both blocks. Here the transitions guards are given next to the edges. The validator checks that the GO and FAIL transitions in the voter state machine given in Figure 11 satisfy the minimum time bound and that the GO transitions occur in a timely manner. Note that the transition from state IGNORE to itself resets the timer T_{cool}.

(15)	$T_{\mathrm{min},h} = \Phi_h^- - \rho$
(16)	$T_{\mathrm{max},h} = \vartheta(\Phi_h^+ + T_{\mathrm{vote}})$
(17)	$T_{\mathrm{vote}} = \vartheta(\sigma + 2d)$
(18)	$T_{\mathrm{idle}} = \vartheta(\sigma + d)$
(19)	$T_{\mathrm{att}} = \vartheta(T_{\mathrm{vote}} + 2d)$
(20)	$\rho = T_{\mathrm{vote}}$
(21)	$T_{\mathrm{cool}} \in \Theta(\max\{\Phi_h^+, \Psi\})$
(22)	$T^* = \max_{h \in \{0,1\}}\{T(\mathbf{A}_h) + 2\Phi_h^+\} + T_{\mathrm{cool}} + \sigma + 2d + \rho$
(23)	$\Phi_h^- > \Psi + 2\rho$
(24)	$\Phi_h^- \geq T_{\mathrm{vote}} + T_{\mathrm{idle}} + T_{\mathrm{att}} + \sigma + 2d$
(25)	$T_{\mathrm{min},h} < T_{\mathrm{cool}}$
(26)	$T_{\mathrm{min},h}/\vartheta > \Psi + \rho$
(27)	$T_{\mathrm{cool}}/\vartheta > 15\beta$
(28)	$C_0 = 4, C_1 = 5$
(29)	$\Lambda_h^+ = T_{\mathrm{max},h} + T_{\mathrm{vote}}$
(30)	$\Lambda_h^- = T_{\mathrm{min},h}/\vartheta$
(31)	$\beta > 2\Psi + 4(T_{\mathrm{vote}} + d) + \rho$
(32)	$\beta \cdot (C_h \cdot j) \leq j \cdot \Lambda_h^- < j \cdot \Lambda_h^+ + \rho \leq \beta \cdot (C_h \cdot j + 1)$ for $0 \leq j \leq 3$

Table 4: The conditions employed in the construction of Section 9. Here $h \in \{0, 1\}$.

We give a detailed analysis of the behaviour of the two state machines later in Sections 9.4.2–9.4.3. The conditions we impose on the timeouts and other parameters are listed in Table 4. As before, system of inequalities listed in Table 4 can be satisfied by choosing the timeouts carefully. This is done in Section 9.4.4.

Lemma 18. *Let $1 < \vartheta < \varphi$ be constants that satisfy $\vartheta^2\varphi < 31/30$. There exists a constant $\Psi_0(\vartheta, \varphi, d)$ such that for any given $\Psi > \Psi_0(\vartheta, \varphi, d)$ we can satisfy the constraints in Table 4 by choosing*

1. *$X \in \Theta(\Psi)$,*
2. *$\Phi_0^- = X$ and $\Phi_0^+ = \varphi X$,*
3. *$\Phi_1^- = rX$ and $\Phi_1^+ = \varphi r X$ for a constant $r > 1$, and*
4. *all remaining timeouts are $O(X)$.*

The resynchronisation algorithm. We now have described all ingredients of our resynchronisation algorithm. It remains to define what is the output of our resynchronisation algorithm. First, for each $h \in \{0, 1\}$, $v \in G$ and $t \geq 0$, let us define an indicator variable for TRIGGER transitions:

$$r_h(u, t) = \begin{cases} 1 & \text{if node } u \text{ transitions to TRIGGER at time } t \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, for each $v \in V_0 \cup V_1$, we define the output of our resynchronisation algorithm as follows:

$$r(v, t) = \max\{r_0(v, t), r_1(v, t)\}.$$

We say that block h *triggers a resynchronisation pulse* at node v if $r_h(v, t) = 1$. That is, node v generates a resynchronisation pulse if either block 0 or block 1 triggers a resynchronisation pulse at node v. Our goal is to show that there exists a time $t \in O(\Phi^+ + \Psi)$ such that every node $v \in G$ generates a resynchronisation pulse at some time $t' \in [t, t + \rho)$ and neither block triggers a new resynchronisation pulse before time $t + \Psi$ at any node $v \in G$. First, however, we observe that the communication overhead incurred by running the voter and validator state machines is small.

Lemma 19. *In order to compute the values $r(v,t)$ for each $v \in G$ and $t \geq 0$, the nodes send at most* $\max\{M(\mathbf{A}_h)\} + O(1)$ *bits per time unit.*

Proof. For both $h \in \{0,1\}$, we have every $v \in V_h$ broadcast a single bit when $\mathbf{A}_h$ generates a pulse locally at node v. Observe that we can assume w.l.o.g. that $\mathbf{A}_h$ generates a pulse only once per time unit even during stabilisation: we can design a wrapper for $\mathbf{A}_h$ that filters out any pulses that occur within e.g. time d of each other. As we have $\Phi_h^- > d$ by Constraint (23), this does not interfere with the pulsing behaviour once the algorithm has stabilised.

In addition to the pulse messages, it is straightforward to verify that the nodes only need to communicate whether they transition to states IDLE, VOTE or PASS in the voter state machine. Due to the T_{vote} and $T_{\max,h}$ timeouts, a node $v \in G$ cannot transition to the same state more than once within d time, as these timeouts are at least ϑd by Constraint (16) and Constraint (17). □

9.3 Proof of Theorem 3

We take a top-down approach for proving Theorem 3. We delay the detailed analysis of the voter and state machines themselves to later sections and now state the properties we need in order to prove Theorem 3.

First of all, as we are considering self-stabilising algorithms, it takes some time for $\mathbf{A}_h$ to stabilise, and hence, also for the voter and validator state machines to start operating correctly. We will show that this is bound to happen by time

$$T^* \in \max\{T(\mathbf{A}_h)\} + O(\max\{\Phi_h^+\} + T_{\text{cool}}) \subseteq \max\{T(\mathbf{A}_h)\} + O(\Psi).$$

The exact value of T^* is given by Constraint (22) and will emerge later in our proofs. We show in Section 9.4.1 that the resynchronisation pulses triggered by a single correct block have skew $\rho = T_{\text{vote}} = \vartheta(\sigma + 2d) \in O(d)$ and the desired separation window of length Ψ; we later argue that a faulty block cannot incessantly interfere with the resynchronisation pulses triggered by the correct block.

Lemma 20 (Stabilisation of correct blocks)**.** *Suppose $h \in \{0,1\}$ is a correct block. Then there exist times $r_{h,0} \in [T^*, T^* + \Phi_h^+ + \rho)$ and, for $j \geq 0$, $r_{h,j+1} \in [r_{h,j} + \Phi^- - \rho, r_{h,j} + \Phi^+ + \rho]$ satisfying the following properties for all $v \in G$ and $j \geq 0$:*

- *$r_h(v,t) = 1$ for some $t \in [r_{h,j}, r_{h,j} + \rho)$,*
- *$r_h(v,t') = 0$ for any $t' \in (t, r_{h,j} + \rho + \Psi)$.*

We also show that if either block $h \in \{0,1\}$ triggers a resynchronisation pulse at some correct node $v \in G$, then for every $u \in G$ (1) a resynchronisation pulse is also triggered by h at roughly the same time or (2) node u refrains from generating a resynchronisation pulse for a sufficiently long time. The latter holds true because node u observes that a resynchronisation pulse *might* have been triggered somewhere (due to the threshold voting mechanism); u thus resets its cooldown counter, that is, transitions to state IGNORE in Figure 11. Formally, this is captured by the following lemma, shown in Section 9.4.2.

Lemma 21 (Grouping lemma for validator machine)**.** *Let $h \in \{0,1\}$ be any block, $t \geq T^*$, and $v \in G$ be such that $r_h(v,t) = 1$. Then there exists $t^* \in [t - 2T_{vote} - d, t]$ such that for all $u \in G$ we have*

$$r_{next}(v,t^*) \in [t^*, t^* + 2(T_{vote} + d)] \cup [t^* + T_{cool}/\vartheta, \infty],$$

where $r_{next}(v,t^*) = \inf\{t' \geq t^* : r_h(v,t') = 1\}$.

Now with the above two lemmas in mind, we take the following proof strategy. Fix a correct block $k \in \{0,1\}$ and let $r_{k,0}$ be the time given by Lemma 20. Observe that if no node $v \in G$ has $r_{1-k}(u,t) = 1$ for any $t \in [r_{k,0}, r_{k,0} + \rho + \Psi)$, then all correct nodes succeed in creating a resynchronisation pulse with separation window Ψ. However, it may be that the other (possibly faulty) block $1-k$ spoils the resynchronisation pulse by also triggering a resynchronisation pulse too soon at some correct node. That is, we may have some $v \in G$ that satisfies $r_{1-k}(u,t) = 1$, where $t \in [r_{k,0}, r_{k,0} + \rho + \Psi)$. But then all nodes observe this and the filtering mechanism now guarantees that the faulty block either obeys the imposed frequency constraints for the following pulses or nodes will ignore them. Either way, we can argue that a correct resynchronisation pulse is generated by the correct block k soon enough.

Accordingly, assume that the faulty block interferes, i.e., generates a spurious resynchronisation pulse at some node $u \in G$ at time $r_{1-k,0} \in (r_{k,0}, r_{k,0} + \rho + \Psi)$. If there is no such time, the resynchronisation pulse of the correct block would have the required separation window of Ψ. Moreover, for all $v \in G$ and $i \geq 0$ we define

$$\begin{aligned} r_{h,0}(v) &= \inf\{t \geq r_{h,0} : r_h(t,v) = 1\} \\ r_{h,i+1}(v) &= \inf\{t > r_{h,i}(v) : r_h(t,v) = 1\}. \end{aligned}$$

Furthermore, for convenience we define the notation

$$D_h(u) = \begin{cases} \emptyset & \text{if block } h \text{ is correct} \\ [r_{h,0}(u) + T_{\text{cool}}/\vartheta, \infty] & \text{otherwise.} \end{cases}$$

For the purpose of our analysis, we "discretise" our time into chunks of length $\beta \in \Theta(\Psi)$; recall that this is similar to what we did in Section 8, where we used chunks of length $\alpha \in \Theta(R)$. For any integers $Y > X \geq 0$, we define

$$\begin{aligned} I_0(X) &= r_{k,0} - 2(T_{\text{vote}} + d) + X \cdot \beta, \\ I_1(X) &= r_{k,0} + 2(T_{\text{vote}} + d) + \Psi + X \cdot \beta, \\ I(X,Y) &= [I_0(X), I_1(Y)) \, . \end{aligned}$$

We abbreviate $\Lambda_h^- = T_{\min,h}/\vartheta$ and $\Lambda_h^+ = T_{\max,h} + T_{\text{vote}}$ (Constraint (29) and Constraint (30)). The following lemma is useful for proving Theorem 3. We defer its proof to Section 9.4.3.

Lemma 22 (Resynchronisation frequency)**.** *For any $j \geq 0$ and $v \in G$ it holds that*

$$r_{h,j}(v) \in \left[r_{h,0} - 2(T_{vote} + d) + j \cdot \Lambda_h^-, r_{h,0} + 2(T_{vote} + d) + j \cdot \Lambda_h^+\right) \cup D_h(v).$$

Corollary 8. *Let $h \in \{0,1\}$ and $0 \leq j \leq 3$. For any $v \in G$ we have*

$$r_{h,j}(v) \in I(j \cdot C_h, j \cdot C_h + 1) \cup D_h(v).$$

Proof. Recall that $r_{1-k,0} \in (r_{k,0} - 2(T_{\text{vote}} + d), r_{k,0} + \rho + \Psi)$. By Constraint (32), for all $j \leq 3$ and $h \in \{0,1\}$ it holds that

$$\beta \cdot C_h \cdot j \leq j \cdot \Lambda_h^- < j \cdot \Lambda_h^+ + \rho \leq \beta \cdot (C_h \cdot j + 1).$$

Using Lemma 22, this inequality, and the above definitions, a straightforward manipulation shows that $r_{h,j}(v)$ lies in the interval

$$\begin{aligned} &[r_{h,0} - 2(T_{\text{vote}} + d) + j \cdot \Lambda_h^-, r_{h,0} + 2(T_{\text{vote}} + d) + j \cdot \Lambda_h^+) \cup D_h(v) \\ \subseteq\ &[r_{k,0} - 2(T_{\text{vote}} + d) + j \cdot \Lambda_h^-, r_{k,0} + 2(T_{\text{vote}} + d) + \Psi + \rho + j \cdot \Lambda_h^+) \cup D_h(v) \\ \subseteq\ &[r_{k,0} - 2(T_{\text{vote}} + d) + \beta \cdot (C_h \cdot j), r_{k,0} + 2(T_{\text{vote}} + d) + \Psi + \beta \cdot (C_h \cdot j + 1)) \cup D_h(v) \\ \subseteq\ &[I_0(j \cdot C_h), I_1(j \cdot C_h + 1)) \cup D_h(v) \\ \subseteq\ &I(j \cdot C_h, j \cdot C_h + 1) \cup D_h(v). \end{aligned}$$

□

With the above results, we can now show that eventually the algorithm outputs a good resynchronisation pulse.

Lemma 23. *There exists a time $t \in \max\{\mathbf{A}_h\} + O(\Psi)$ such that for all $v \in G$ there exists a time $t' \in [t, t + \rho]$ satisfying*

- *$r(v, t') = 1$, and*
- *$r(v, t'') = 0$ for $h \in \{0, 1\}$ and any $t'' \in (t', t' + \Psi)$.*

Proof. Suppose block $k \in \{0, 1\}$ is correct. The lemma follows by proving that we have the following properties for some $t \le I_1(11)$ and each $v \in G$:

- $r_k(v, t') = 1$ for some $t' \in [t, t + \rho]$, and
- $r_h(v, t'') = 0$ for $h \in \{0, 1\}$ and any $t'' \in (t', t' + \Psi)$.

Recall that $r_{1-k,0} \in (r_{k,0} - 2(T_{\text{vote}} + d), r_{k,0} + \rho + \Psi)$, as otherwise the claim trivially follows for $t = r_{k,0}$. Consider any $v \in G$. Corollary 8 and the fact that $C_0 = 4$ by Constraint (28) implies

$$\begin{aligned} r_{0,0}(v) &\in I(0,1) \cup D_0(v) \\ r_{0,1}(v) &\in I(4,5) \cup D_0(v) \\ r_{0,2}(v) &\in I(8,9) \cup D_0(v) \\ r_{0,3}(v) &\in I(12,13) \cup D_0(v). \end{aligned}$$

As $T_{\text{cool}}/\vartheta \ge 15\beta$ by Constraint (27), it follows for all $t \ge r_{0,0}$ that if $r_0(v, t) = 1$, then

$$t \in I(0,1) \cup I(4,5) \cup I(8,9) \cup [I_0(12), \infty].$$

Similarly, as $C_1 = 5$, for all $t \ge r_{1,0}$ we get that if $r_1(v, t) = 1$, then

$$t \in I(0,1) \cup I(5,6) \cup I(10,11) \cup [I_0(15), \infty].$$

Let k be the correct block we have fixed. Recall that $D_k(v) = \emptyset$. The claim now follows from a simple case analysis:

1. If $k = 0$, then $r_{k,2}(v) \in I(8,9)$ and $r_{1-k}(v, t') = 0$ for all $t' \in [I_1(6), I_0(10)) \supset [I_0(8), I_1(9) + \Psi + \rho)$ (by Constraint (31)).
2. If $k = 1$, then $r_{k,2}(v) \in I(10,11)$ and $r_{1-k}(v, t') = 0$ for all $t' \in [I_1(9), I_0(12)) \supset [I_0(10), I_1(11) + \Psi + \rho)$ (by Constraint (31)).

Thus, in both cases $t = \min_{v \in G}\{r_{k,2}(v)\}$ satisfies the claim of the lemma, provided that $t \le I_1(11) \in \max\{\mathbf{A}_h\} + O(\Psi)$. This is readily verified from the constraints given in Table 4. □

Theorem 3. *Let $f, n_0, n_1 \in \mathbb{N}$, $n = n_0 + n_1$, $f_0 = \lfloor (f-1)/2 \rfloor$, $f_1 = \lceil (f-1)/2 \rceil$, and $1 < \vartheta \le 1.007$. Suppose that for some given $\Psi \in \Omega(1)$, sufficiently small constant $\varphi > \varphi_0(\vartheta)$, and $T_0 \in \Theta(\Psi)$, it holds that for any $h \in \{0, 1\}$ and $T_0 \le T \in O(\Psi)$ there exists a pulse synchronisation algorithm $\mathbf{A}_h$ that*

- *runs on n_h nodes and tolerates f_h faulty nodes,*
- *has skew $\sigma = 2d$ and accuracy bounds $\Phi_h^- = T$ and $\Phi_h^+ = T\varphi$.*

Then there exists a resynchronisation algorithm with skew $\rho \in O(d)$ and separation window of length Ψ that generates a resynchronisation pulse by time $\max\{T(\mathbf{A}_0), T(\mathbf{A}_1)\} + O(\Psi)$, where nodes broadcast only $O(1)$ additional bits per time unit.

Proof. Computation shows that for $\vartheta \leq 1.007$, we have that $\vartheta^2 \phi_0(\vartheta) < 31/30$ (where $\phi_0(\vartheta) = 1 + 6(\vartheta - 1)/(2 + 3\vartheta - 4\vartheta^2)$ is given in Corollary 7). Thus, Lemma 18 shows that for a sufficiently small choice of $\varphi > \varphi_0(\vartheta) > \vartheta$ we can pick $\Phi_h^- \in \Theta(\Psi)$, where $1 < \max\{\Phi_h^+/\Phi_h^-\} \leq \varphi$, such that the conditions given in Table 4 are satisfied. Thus, using our assumption, we can choose the algorithms $\mathbf{A}_h$ with these accuracy bounds Φ_h; note that here we use sufficiently small ϑ and φ that satisfy both our initial assumption and the preconditions of Lemma 18.

In order to compute the output value $r(v,t) \in \{0,1\}$ for each $v \in G$ and $t \geq 0$, Lemma 19 shows that our resynchronisation algorithm only needs to communicate $O(1)$ bits per time unit in addition to the message sent by underlying pulse synchronisation algorithms $\mathbf{A}_0$ and $\mathbf{A}_1$. By Lemma 23, we have that a good resynchronisation pulse with skew $\rho \in O(d)$ happens at a time $t \in \max\{\mathbf{A}_h\} + O(\Psi)$. □

9.4 Proofs of the remaining lemmas

We now give proofs of the missing lemmas.

9.4.1 Proof of Lemma 20

We now show that eventually a correct block $h \in \{0,1\}$ will start triggering resynchronisation pulses with accuracy bounds $\Lambda_h = (\Lambda_h^-, \Lambda_h^+)$. Our first goal is to show that after the algorithm $\mathbf{A}_h$ has stabilised in a correct block h, all correct nodes will start transitioning to GO in a synchronised fashion. Then we argue that eventually the transitions to the state GO will be coupled with the transitions to TRIGGER.

Recall that the pulse synchronisation algorithm $\mathbf{A}_h$ has skew σ and accuracy bounds $\Phi_h = (\Phi_h^-, \Phi_h^+)$. Let $p_h(v,t) \in \{0,1\}$ indicate whether node $v \in V_h \setminus F$ generates a pulse according to algorithm $\mathbf{A}_h$ at time t. If block $h \in \{0,1\}$ is corrrect, then by time $T(\mathbf{A}_h)$ the algorithm $\mathbf{A}_h$ has stabilised. Moreover, then there exists a time $T(\mathbf{A}_h) \leq p_{h,0} \leq T(\mathbf{A}_h) + \Phi_h^+$ such that each $v \in V_h \setminus F$ satisfies $p_h(v,t) = 1$ for some $t \in [p_{h,0}, p_{h,0} + \sigma)$. Since block h and algorithm $\mathbf{A}_h$ are correct, there exist for each $v \in V_h \setminus F$ and $i \geq 0$ the following values:

$$\begin{aligned} p_{h,i}(v) &= \inf\{t \geq p_{h,i} : p_h(v,t) = 1\} \neq \infty, \\ p_{h,i+1} &\in [p_{h,i} + \Phi_h^-, p_{h,i} + \Phi_h^+), \\ p_{h,i+1}(v) &\in [p_{h,i+1}, p_{h,i+1} + \sigma). \end{aligned}$$

That is, $\mathbf{A}_h$ generates a pulse at node $v \in V_h$ for the ith time after stabilisation at time $p_{h,i}(v)$.

First, let us observe that the initial "clean" pulse makes every correct node transition to GO or FAIL, thereby resetting the $T_{\max,h}$ timeouts, where the nodes will wait until the next pulse.

Lemma 24. *Suppose block $h \in \{0,1\}$ is correct. Each correct node $v \in G$ is in state* IDLE *at time $p_{h,1}$ and its local $T_{max,h}$ timer does not expire during the interval $[p_{h,1}, p_{h,1} + \sigma + d)$.*

Proof. First observe that if the timer $T_{\max,h}$ is reset at time $p_{h,0}$ or later, then it will not expire before time $p_{h,0} + T_{\max,h}/\vartheta > p_{h,0} + \Phi_h^+ + \sigma + d \geq p_{h,1} + \sigma + d$ by Constraint (16) and Constraint (17). Because every node receives $n_h - f_h$ pulse messages from different nodes in V_h during $(p_{h,0}, p_{h,0} + \sigma + d)$ and $T_{\text{idle}} = \vartheta(\sigma + d)$ by Constraint (18), every node that is in state IDLE at time p_{h_0} transitions leaves this state during $(p_{h_0}, p_{h_0} + \sigma + d)$. Recall that nodes cannot stay in states LISTEN, GO, or FAIL. Because nodes leave states AWAKE, VOTE, and PASS when the timeout $\langle T_{\text{vote}}, \text{LISTEN}\rangle$ expires, any node not in state IDLE must transition to this state within T_{vote} time. Accordingly, each correct node resets its $T_{\max,h}$ timer during $(p_{h,0}, p_{h,0} + T_{\text{vote}} + \sigma + d)$.

Next, note that during $(p_{h_0} + T_{\text{idle}} + \sigma + d, p_{h_1})$, no correct node has any pulse messages from correct nodes in V_h in its respective buffer. Therefore, no correct node can transition to vote during this time interval. This, in turn, implies that no correct node has any VOTE messages from correct nodes in its respective buffer with timeout T_{att} during $(p_{h_0} + T_{\text{idle}} + T_{\text{att}} + \sigma + 2d, p_{h_1})$.

Therefore, no correct node can leave state IDLE during this period. Finally, any correct nodes not in state IDLE at time $p_{h_0} + T_{\text{idle}} + T_{\text{att}} + \sigma + 2d$ must transition back to IDLE by time $p_{h_0} + T_{\text{vote}} + T_{\text{idle}} + T_{\text{att}} + \sigma + 2d$. As $T_{\text{vote}} + T_{\text{idle}} + T_{\text{att}} + \sigma + 2d \leq \Phi^-$ by Constraint (24), the claim follows. □

Let us define an indicator variable for GO transitions:

$$g_h(v,t) = \begin{cases} 1 & \text{if node } u \text{ transitions to GO at time } t \\ 0 & \text{otherwise.} \end{cases}$$

Similarly to above, we now also define for $v \in G$ and $i \geq 1$ the values

$$\begin{aligned} g_{h,1} &= \inf\{t \geq p_1 : g_h(u,t) = 1, u \in G\}, \\ g_{h,1}(v) &= \inf\{t \geq g_{h,1} : g_h(v,t) = 1\}, \\ g_{h,i+1}(v) &= \inf\{t > g_{h,i}(v)\}. \end{aligned}$$

In words, the time $g_{h,1}$ is the minimal time that some correct node transitions to state GO in the voter state machine of block h at or after the second pulse of $\mathbf{A}_h$ after stabilisation. The two other values indicate the ith time a correct node $v \in G$ transitions to GO starting from time $g_{h,1}$.

We now show that starting from the second pulse $p_{h,1}$ of a correct block h, the GO signals essentially just "echo" the pulse.

Lemma 25. *If block h is correct, then for all $v \in G$ and $i > 0$ it holds that*

$$g_{h,i}(v) \in (p_{h,i} + \sigma + 2d, p_{h,i} + \sigma + 2d + \rho),$$

where $\rho = \vartheta(\sigma + 2d) = T_{vote}$. Moreover, node v does not transition to state FAIL *at any time $t \geq p_{h,1}$.*

Proof. By Lemma 24 we have that each $v \in G$ is in state IDLE at time $p_{1,h}$. During $(p_{h,1}, p_{h,1} + \sigma + d)$, i.e., within T_{idle} local time by Constraint (18), each node receives $n_h - f_h$ pulse messages from different nodes in V_h and thus transitions to VOTE. Thus, all nodes receive $n - f$ VOTE messages from different nodes during $(p_{h,1}, p_{h,1} + \sigma + 2d)$. As $T_{\text{vote}}/\vartheta = \sigma + 2d$ by Constraint (17), each correct node transitions to PASS before $\langle T_{\text{vote}}, \text{LISTEN}\rangle$ expires and transitions to GO at a time from $(p_{h,1} + \sigma + 2d, p_{h,1} + \sigma + 2d + T_{\text{vote}}) = (p_{h,1} + \sigma + 2d, p_{h,1} + \sigma + 2d + \rho)$. In particular, it resets its buffers after all pulse and VOTE messages from correct nodes are received. Consequently, correct nodes stay in state IDLE until the next pulse, which occurs at time $p_{h,2} > p_{h,1} + T_{\text{att}} + \sigma + 2d > p_{h,1} + T_{\text{vote}} + \sigma + 2d$ by Constraint (19) and Constraint (24). Repeating this reasoning inductively completes the proof. □

We define the following time bound for each $h \in \{0,1\}$:

$$T_h^* = T(\mathbf{A}_h) + T_{\text{cool}} + 2\Phi_h^+ + \sigma + 2d + \rho.$$

We now show that by time T_h^* we are guaranteed that transitions to GO and TRIGGER have become coupled if block h is correct.

Lemma 26. *Suppose block $h \in \{0,1\}$ is correct. Then for any $v \in G$ and $t \geq T_h^*$ it holds that*

$$r_h(v,t) = 1 \textit{ if and only if } g_h(v,t) = 1.$$

Proof. Note that node $u \in G$ can transition to state TRIGGER from WAIT in the validator state machine only if the voter state machine transitions to GO. Consider the time $g_{h,j}(v)$ for $j > 0$. Observe that there are three states in which v may be at this time: WAIT, HOLD, or IGNORE. We argue that in each case node v eventually is in state WAIT in the validator state machine when the voter state machine transitions to state GO, and thus, node v transitions to TRIGGER.

First of all, note that by Lemma 25 node v does not transition to the state FAIL at any time $t' \geq g_{h,j}(v) \geq p_{h,1}$. We utilise this fact in each of the three cases below:

1. In the first case, node v transitions from WAIT to TRIGGER at time $g_{h,j}(v)$ and hence we have $r_h(v, g_{h,j}(v)) = 1$. By applying both Lemma 25 and Constraint (15) we get that $g_{h,j+1}(v) \geq g_{h,j}(v) + \Phi_h^- \geq g_{h,j}(v) + T_{\min,h}$. Moreover, v does not transition to the FAIL state in the voter state machine at any time $t' \geq g_{h,j}$. Hence, by induction, node v transitions from state WAIT to TRIGGER at time $g_{h,j'}(v)$ for each $j' \geq j$.

2. In the second case, v transitions from HOLD to IGNORE at time $g_{h,j}(v)$. By time $r \leq g_{h,j}(v) + T_{\text{cool}}$ node v transitions to WAIT. Hence, for any j' with $g_{h,j'}(v) \geq r$, the first case applies.

3. In the third case, v remains in state IGNORE until at a time $r \leq g_{h,j}(v) + T_{\text{cool}}$ its T_{cool} timer expires and v transitions to WAIT. Again, for any j' with $g_{h,j'}(v) \geq r$, the first case applies.

Now consider the time $g_{h,1}(v) \in [p_{h,1} + \sigma + 2d, p_{h,1} + \sigma + 2d + \rho)$ given by Lemma 25. From the above case analysis we get that node v is in state WAIT by time

$$g_{h,1}(v) + T_{\text{cool}} < p_{h,1} + \sigma + 2d + \rho + T_{\text{cool}} \leq T_h^*,$$

and from then on each transition to GO entails a transition to TRIGGER, as claimed. □

Lemma 20 (Stabilisation of correct blocks)**.** *Suppose $h \in \{0, 1\}$ is a correct block. Then there exist times $r_{h,0} \in [T^*, T^* + \Phi_h^+ + \rho)$ and, for $j \geq 0$, $r_{h,j+1} \in [r_{h,j} + \Phi^- - \rho, r_{h,j} + \Phi^+ + \rho]$ satisfying the following properties for all $v \in G$ and $j \geq 0$:*

- $r_h(v, t) = 1$ *for some* $t \in [r_{h,j}, r_{h,j} + \rho)$,
- $r_h(v, t') = 0$ *for any* $t' \in (t, r_{h,j} + \rho + \Psi)$.

Proof. First, observe that by Constraint (22) we have $T^* = \max\{T_h^*\}$. Let $p_{h,i} \in [T^* - \sigma - 2d, T^* - \sigma - 2d + \Phi_h^+]$ for some $i > 0$. By Lemma 25, we have that for all $i' \geq i$ and $v \in G$ it holds that

$$g_{h,i'}(v) \in (p_{h,i'} + \sigma + 2d, p_{h,i'} + \sigma + 2d + \rho),$$

and by Lemma 26, we have $r_h(v, t) = 1$ for all $g_{h,i'}(v) = t \geq T_h^*$ and $r_h(v, t) = 0$ for all other $t \geq T^*$. We set $r_{h,j} = \min_{v \in G}\{g_{h,i+j}(v)\}$. As $p_{h,i'+1} - p_{h,i'} \in [\Phi^-, \Phi^+]$ for all $i' \geq 0$, this shows all required time bounds but $r_h(v, t') = 0$ for each $v \in G$, j, and $t' \in (g_{h,i+j}(v), r_{h,j} + \rho + \Psi)$. The latter follows because $\Phi^- > \Psi + 2\rho$ by Constraint (23). □

9.4.2 Proof of Lemma 21

Lemma 21 (Grouping lemma for validator machine)**.** *Let $h \in \{0, 1\}$ be any block, $t \geq T^*$, and $v \in G$ be such that $r_h(v, t) = 1$. Then there exists $t^* \in [t - 2T_{vote} - d, t]$ such that for all $u \in G$ we have*

$$r_{next}(v, t^*) \in [t^*, t^* + 2(T_{vote} + d)] \cup [t^* + T_{cool}/\vartheta, \infty],$$

where $r_{next}(v, t^*) = \inf\{t' \geq t^* : r_h(v, t') = 1\}$.

In order to show Lemma 21, we analyse how the voting and validator state machines given in Figure 11 and Figure 12 behave. We show that the voter machines for a single block $h \in \{0, 1\}$ are roughly synchronised in the following sense: if some correct node transitions to GO, then every correct node will transition to either GO or FAIL within a short time window.

Lemma 27. *Let $t \geq 2T_{vote} + d$ and $v \in G$ such that $g_h(v, t) = 1$. Then there exists $t^* \in (t - 2T_{vote} - d, t]$ such that all correct nodes transition to* GO *or* FAIL *during the interval $[t^*, t^* + 2(T_{vote} + d))$.*

Proof. Note that at any time $t' \geq T_{\text{vote}} + d$, any VOTE message stored at a correct node (supposedly) sent by another correct node must actually have been sent at a time greater than 0. Since $v \in G$ satisfies $g_h(v,t) = 1$, this means that there is a time $t' \in [t - T_{\text{vote}}, t]$ such that v received at least $n - 2f$ different VOTE messages from correct nodes during the interval $[t' - T_{\text{vote}}, t']$. Hence, every correct $u \in G$ must receive at least $n - 2f > f$ VOTE messages from different nodes during the interval $I = (t' - T_{\text{vote}} - d, t' + d)$.

Let $t^* < t'$ be the minimal time a correct node transitions to VOTE during the interval I. Consider any node $u \in G$. If u does not clear its VOTE memory flags (by transitioning to IDLE) during the interval I, by the above observations u has memorised at least $f + 1$ VOTE messages in the buffer using timeout T_{att} at some time $t'' \in [t^*, t' + d]$ and must transition to LISTEN in case it is in state IDLE. However, any node transitioning to LISTEN will transition to GO or FAIL within T_{vote} time. Overall, each correct node must transition to FAIL of GO during the interval $[t^*, t' + T_{\text{vote}} + d] \subseteq [t^* + 2(T_{\text{vote}} + d)]$. □

We now show a similar synchronisation lemma for the validator state machines as well: if some correct node transitions to TRIGGER and triggers a resynchronisation pulse, then every correct node triggers a resynchronisation pulse or transition to IGNORE within a short time window.

Lemma 28. *Let $t \geq 2T_{vote} + d$ and suppose $r_h(u,t) = 1$ for some $v \in G$. Then there exists a time $t^* \in (t - 2T_{vote} - d, t]$ such that all correct nodes transition to* TRIGGER *or* IGNORE *during the time interval $[t^*, t^* + 2(T_{vote} + d))$.*

Proof. Suppose some node v transitions to state TRIGGER at time t. Then it must have transitioned to state GO in the voter state machine at time t as well. By Lemma 27 we get that there exists t^* such that all correct nodes transition to GO or FAIL during the interval $[t^*, t^* + 2(T_{\text{vote}} + d))$. Once $u \in G$ transitions to either of these states in the voter state machine, this causes a transition in the validator state machine to either TRIGGER or IGNORE, as can be seen from Figure 12. Hence, during the same interval, all correct nodes will either transition to TRIGGER or IGNORE. □

Observe that once $v \in G$ transitions to IGNORE at time t, then it cannot transition back to TRIGGER before time T_{cool} time has passed on its local clock, that is, before time $t + T_{\text{cool}}/\vartheta$. Thus, Lemma 28 now implies Lemma 21, as $T^* \geq 2T_{\text{vote}} + d$.

9.4.3 Proof of Lemma 22

We now aim to prove Lemma 22. Hence, let $r_{h,0}$ be as defined in Section 9.3. We have shown above that if block h is correct, then the resynchronisation pulses generated by block h are coupled with the pulses generated by the underlying pulse synchronisation algorithm $\mathbf{A}_h$. We will argue that any block $h \in \{0, 1\}$, including a faulty one, must either respect the accuracy bounds $\Lambda_h = (\Lambda_h^-, \Lambda_h^+)$ when triggering resynchronisation pulses or refrain from triggering a resynchronisation pulse for at least time $T_{\text{cool}}/\vartheta$.

First, we note that Lemma 25 and Lemma 26 together imply the following corollary.

Corollary 9. *Suppose block $h \in \{0, 1\}$ is correct. For all $u \in G$ and $i \geq 0$ we get that*

$$r_{h,i+1}(u) \in [r_{h,i}(u) + \Lambda_h^-, r_{h,i}(u) + \Lambda_h^+).$$

Proof. By Lemma 20, we get that

$$r_{h,i+1}(u) \in [r_{h,i}(u) + \Phi_h^- - \rho, r_{h,i}(u) + \Phi_h^+ + \rho).$$

To see the lower bound holds, note that $\Lambda_h^- = T_{\min,h}/\vartheta = \Phi_h^- - \rho$ by Constraint (30) and Constraint (15). For the upper bound, we have from Constraint (29) that $\Lambda_h^+ = T_{\max,h} + T_{\text{vote}}$. Since $T_{\max,h} > \Phi_h^+ + T_{\text{vote}} = \Phi_h^+ + \rho$ by Constraint (16) and Constraint (20), the claim follows. □

Lemma 29. *Let $u \in G$, $h \in \{0,1\}$, and $i \geq 0$. Then*

$$r_{h,i+1}(u) \in [r_{h,i}(u) + \Lambda_h^-, r_{h,i}(u) + \Lambda_h^+] \cup D_h(u).$$

Proof. First observe that in case $r_{h,i}(u) = \infty$ for any $i \geq 0$, by definition $r_{h,i+1}(u) = \infty \in D_h(u)$ and the claim holds.

Hence, let $t = r_{h,i}(u) \neq \infty$. Since u transitions to TRIGGER at time t, it follows that u also transitions to state GO at time t, that is, $g_h(u,t) = 1$. Therefore, u will transition to state IDLE in the voter state machine and state HOLD in the validator state machine. Observe that u cannot transition to TRIGGER again before time $t + \min\{T_{\min,h}, T_{\text{cool}}\}/\vartheta$, that is, before either local timer $T_{\min,h}$ or T_{cool} expires. Since $T_{\min,h} < T_{\text{cool}}$ by Constraint (25), we get that $r_{h,i+1}(u) \geq t + T_{\min,h}/\vartheta = t + \Lambda_h^-$.

Next note that u transitions to FAIL when (1) the local timer $T_{\max,h}$ expires when in state IDLE or (2) T_{vote} expires when not in state IDLE. By time $t + T_{\max,h}$ node u has to transition from state IDLE to LISTEN, and from there to either FAIL or GO via VOTE, AWAKE, and/or PASS. This implies that by the time $t + T_{\max,h} + T_{\text{vote}}$ node u has transitioned to either TRIGGER or IGNORE in the validator state machine. Hence, we get that $r_{h,i+1}(u) \leq t + T_{\max,h} + T_{\text{vote}} = r_{h,i}(u) + \Lambda_h^+$ or $r_{h,i+1}(u) \geq r_{h,i}(u) + T_{\text{cool}}/\vartheta \geq r_{h,0}(u) + T_{\text{cool}}/\vartheta$. Therefore, the claim follows. □

Lemma 22 (Resynchronisation frequency)**.** *For any $j \geq 0$ and $v \in G$ it holds that*

$$r_{h,j}(v) \in \left[r_{h,0} - 2(T_{vote} + d) + j \cdot \Lambda_h^-, r_{h,0} + 2(T_{vote} + d) + j \cdot \Lambda_h^+\right) \cup D_h(v).$$

Proof. If block h is correct, Corollary 9 shows the claim, so assume that h is faulty. Again observe that if $r_{h,j}(v) = \infty$, then the claim vacuously holds for all $j' \geq j$. We prove the claim by induction on increasing j, so w.l.o.g. we may assume that $r_{h,j} \neq \infty$ for all $j \geq 0$. The base case $j = 0$ follows directly from the definition of $r_{h,0}$ and Lemma 21. By applying Lemma 29 to index j, v, and h, and using the induction hypothesis we get that $r_{h,j+1}(v)$ lies in the interval

$$\begin{aligned}
&[r_{h,j}(v) + \Lambda_h^-, r_{h,j}(v) + \Lambda_h^+] \cup [r_{h,j}(v) + T_{\text{cool}}/\vartheta, \infty] \\
\subseteq\, &[r_{h,0} - 2(T_{\text{vote}} + d) + (j+1) \cdot \Lambda_h^-, r_{h,0} + 2(T_{\text{vote}} + d) + (j+1) \cdot \Lambda_h^+) \cup D_h(v).
\end{aligned}$$
□

9.4.4 Proof of Lemma 18

Lemma 18. *Let $1 < \vartheta < \varphi$ be constants that satisfy $\vartheta^2\varphi < 31/30$. There exists a constant $\Psi_0(\vartheta, \varphi, d)$ such that for any given $\Psi > \Psi_0(\vartheta, \varphi, d)$ we can satisfy the constraints in Table 4 by choosing*

1. *$X \in \Theta(\Psi)$,*
2. *$\Phi_0^- = X$ and $\Phi_0^+ = \varphi X$,*
3. *$\Phi_1^- = rX$ and $\Phi_1^+ = \varphi r X$ for a constant $r > 1$, and*
4. *all remaining timeouts are $O(X)$.*

Proof. Let $1 < \vartheta < \varphi$ be constants that satisfy $\vartheta^2\varphi < 31/30$. We show that we can satisfy the constraints by setting

$$\begin{aligned}
\Phi_0^- &= X \\
\Phi_0^+ &= \varphi X \\
\Phi_1^- &= rX \\
\Phi_1^+ &= r\varphi X \\
\Psi &= aX \\
\beta &= bX \\
T_{\text{cool}} &= cX,
\end{aligned}$$

where $a = b/3$, $b = 6/25 \cdot \vartheta\varphi$, $c = 16\vartheta b$, $r = 31/25$ and by picking a sufficiently large $X > X_0(\vartheta, \varphi, d)$. Here $X_0(\vartheta, \varphi, d)$ depends only on the given constants. Note that the choice of T_{cool} satisfies Constraint (21).

First, let us pick the values for the remaining timeouts and T^* as given by Constraints (15)–(20), (28)–(30), and (22); it is easy to check that these equalities can be satisfied simultaneously. Regarding Constraint (23), observe that $3/25 \cdot \vartheta\varphi < 1$ and

$$\Phi_h^- \geq X > 3/25 \cdot \vartheta\varphi X + 2\rho = aX + 2\rho = \Psi + 2\rho$$

when $X > 2\rho/(1 - 3/25 \cdot \vartheta\varphi)$, that is, X is larger than a constant. Furthermore, Constraint (24) is also satisfied by picking the constant bounding X from below to be large enough.

To see that Constraint (25) holds, observe that $T_{\text{cool}} = cX = 16\vartheta^2\rho X \cdot 6/25 > 96/25 \cdot X > 31/25 \cdot X = rX \geq \Phi_h^-$ for both $h \in \{0, 1\}$. Assuming $X > 2\rho \cdot 375/344 = 2\rho/(1 - 2/25 \cdot 31/30)$, Constraint (26) is satisfied since

$$\Phi_h^- - \rho \geq X - \rho > 2/25 \cdot 31/30 \cdot X + \rho > 2/25 \cdot \vartheta^2\varphi X + \rho > b/3 \cdot X + \rho = aX + \rho = \Psi + \rho.$$

Constraint (27) is satisfied as $T_{\text{cool}}/\vartheta = cX/\vartheta = 16bX = 16\beta > 15\beta$. Having $X > 3/b \cdot (5\rho + 4d)$ yields that Constraint (31) is satisfied, since

$$2\Psi + 4(T_{\text{vote}} + d) + \rho = 2\Psi + 5\rho + 4d = 2b/3 \cdot X + 5\rho + 4d < bX = \beta.$$

It remains to address Constraint (32). As Constraint (15) and Constraint (30) hold, the first inequality of Constraint (32) is equivalent to

$$\vartheta\beta C_h = 6/25 \cdot \vartheta^2\varphi X C_h \leq \Phi_h^- - \rho.$$

We have $C_0 = 4$ by Constraint (28). For $X > \rho/(1 - 24/25 \cdot \vartheta^2\varphi)$,

$$24/25 \cdot \vartheta^2\varphi X < X - \rho = \Phi_0^- - \rho$$

thus shows that the inequality holds. For $h = 1$, $C_1 = 5$. Recalling that $\vartheta^2\varphi < 31/30$, we may assume that $X > \rho/(31/25 - 6/5 \cdot \vartheta^2\varphi)$, yielding

$$6/5 \cdot \vartheta^2\varphi X < 31/25 \cdot X - \rho = rX - \rho = \Phi_1^- - \rho,$$

i.e., the first inequality of Constraint (32) is satisfied for $h = 1$. The middle inequality is trivially satisfied as $\Lambda_h^- < \Lambda_h^+$. By the already established equalities, the final inequality in Constraint (32) is equivalent to

$$j\vartheta\Phi_h^+ + ((\vartheta + 1)j + 1)\rho \leq \beta(C_h \cdot j + 1)$$

for all $h \in \{0, 1\}$ and $0 \leq j \leq 3$.

Let $A_j = ((\vartheta + 1)j + 1)\rho$ and observe that $25/3 \cdot A_3 > 25/4 \cdot A_2 > 5A_1$. For any $X > 25/3 \cdot A_3$ and $h = 0$, a rudimentary calculation thus shows

$$\begin{aligned}
\vartheta\varphi X + A_1 &\leq 30/25 \cdot \vartheta\varphi X = 5bX \\
2\vartheta\varphi X + A_2 &\leq 54/25 \cdot \vartheta\varphi X = 9bX \\
3\vartheta\varphi X + A_3 &\leq 78/25 \cdot \vartheta\varphi X = 13bX.
\end{aligned}$$

Since $\Phi_0^+ = \varphi X$, $\beta = bX$, and $C_0 = 4$, this covers the case of $h = 0$. Similarly, as $r = 31/25 = 1 + b/(\vartheta\varphi)$, we have

$$\begin{aligned}
\vartheta\varphi r X + A_1 &\leq 6bX \\
2\vartheta\varphi r X + A_2 &\leq 11bX \\
3\vartheta\varphi r X + A_3 &\leq 16bX,
\end{aligned}$$

covering the case of $h = 1$. Overall, we conclude that Constraint (32) is satisfied.

Finally, observe that in all cases we assumed that X is lower bounded below by a function $X_0(\vartheta, \varphi, d)$ which depends only on the constants ϑ, φ, and message delay d. Thus, the constraints can be satisfied by picking $X > X_0(\vartheta, \varphi, d)$ which yields that we can satisfy the constraints for any $\Psi > \Psi_0(\vartheta, \varphi, d) = aX_0(\vartheta, \varphi, d)$. □

10 Randomised algorithms

While we have so far only considered deterministic algorithms, our framework also extends to randomised algorithms. In particular, this allows us to obtain faster algorithms by simply replacing the synchronous consensus algorithms used with randomised variants. Randomised consensus algorithms can break the linear-time lower bound [20] for deterministic algorithms [22, 31]. This in turn allows us to construct the first pulse synchronisation algorithms that stabilise in sublinear time.

Typically, when considering randomised consensus, one relaxes the termination property: it suffices that the algorithm terminates with probability 1 and gives probabilistic bounds on the (expected, w.h.p., etc.) round complexity. However, our framework operates based on a deterministic termination guarantee, where the algorithm is assumed to declare its output in R rounds. Therefore, we instead relax the *agreement* property so that it holds with a certain probability only. Formally, node v is given an input $x(v) \in \{0,1\}$, and it must output $y(v) \in \{0,1\}$ such that the following properties hold:

1. **Agreement:** With probability at least p, there exists $y \in \{0,1\}$ such that $y(v) = y$ for all correct nodes v.

2. **Validity:** If for $x \in \{0,1\}$ it holds that $x(v) = x$ for all correct nodes v, then $y(v) = x$ for all correct nodes v.

3. **Termination:** All correct nodes decide on $y(v)$ and terminate within R rounds.

This modification is straightforward, as the following lemma shows.

Lemma 30. *Let* $\mathbf{C}$ *be a randomised synchronous consensus routine that terminates in* R *rounds in expectation and deterministically satisfies agreement and validity conditions. Then there exists a randomised synchronous consensus routine* $\mathbf{C}'$ *that deterministically satisfies validity and terminates within* $2R$ *rounds, and satisfies agreement with probability at least* $1/2$*. All other properties, such as message size and resilience, of* $\mathbf{C}$ *and* $\mathbf{C}'$ *are the same.*

Proof. The modified algorithm operates as follows. We run the original algorithm for (up to) $2R$ rounds. If it terminates at node v, then node v outputs the decision of the algorithm. Otherwise, node v outputs its input value so that $y(v) = x(v)$. This deterministically guarantees validity: if all correct nodes have the same input, the original algorithm can only output that value. Concerning agreement, observe that by Markov's bound, the original algorithm has terminated at all nodes within $2R$ rounds with probability at least $1/2$. Accordingly, agreement holds with probability at least $1/2$. □

We remark that the construction from [26] that generates silent consensus routines out of regular ones also applies to randomised algorithms (as produced by Lemma 30), that is, we can obtain suitable randomised silent consensus routines to be used in our framework.

Our framework makes use of consensus in the construction underlying Theorem 2 only. For stabilisation, we need a constant number of consecutive consensus instances to succeed. Thus, a constant probability of success for each individual consensus instance is sufficient to maintain an expected stabilisation time of $O(R)$ for each individual level in the stabilisation hierarchy. This is summarised in the following variant of Theorem 2.

Corollary 10. *Let* $f \geq 0$ *and* $n > 3f$*. Suppose for a network of* n *nodes there exist*

- *an* f*-resilient resynchronisation algorithm* $\mathbf{B}$ *with skew* $\rho \in O(d)$ *and separation window* $\Psi \geq \Psi_0$ *for a sufficiently large* $\Psi_0 \in O(R)$ *and*
- *an* f*-resilient randomised synchronous consensus algorithm* $\mathbf{C}$*,*

where $\mathbf{C}$ runs in $R = R(f)$ rounds, lets nodes send at most $M = M(f)$ bits per round, and validity and agreement hold with constant probability. Then there exists a randomised f-resilient pulse synchronisation algorithm $\mathbf{A}$ for n nodes with skew $\sigma = 2d$ and accuracy bounds $\Phi^-, \Phi^+ \in \Theta(R)$ that stabilises in expected $T(\mathbf{B}) + O(R)$ time and has nodes send $M(\mathbf{B}) + O(M)$ bits per time unit.

Note that once one of the underlying pulse synchronisation algorithms used in the construction of Theorem 3 stabilise, the resynchronisation algorithm itself stabilises deterministically, as it does not make use of consensus or randomisation. Applying linearity of expectation and the same recursive pattern as before, Theorem 1 thus generalises as follows.

Corollary 11. *Let $\mathcal{C}$ be a family of randomised synchronous consensus routines satisfying that (i) for any $f_0, f_1 \in \mathbb{N}$, $n(f_0 + f_1) \leq n(f_0) + n(f_1)$, (ii) both $M(f)$ and $R(f)$ are increasing, and (iii) agreement holds with constant probability. Then, for any $f \geq 0$, $n \geq n(f)$, and $T \geq T_0$ for some $T_0 \in \Theta(R(f))$, there exists a T-pulse synchronisation algorithm $\mathbf{A}$ with skew $2d$. Moreover, $\mathbf{A}$ stabilises in expected time*

$$T(\mathbf{A}) \in O\left(d + \sum_{k=0}^{\lceil \log f \rceil} R(2^k)\right)$$

and has nodes broadcast

$$M(\mathbf{A}) \in O\left(1 + \sum_{k=0}^{\lceil \log f \rceil} M(2^k)\right),$$

bits per time unit, where the sums are empty when $f = 0$.

However, while randomised algorithms can be more efficient, typically they require additional restrictions on the model, e.g., that the adversary must not be able to predict future random decisions. Naturally, such restrictions then also apply when applying Corollary 10 and, subsequently, Corollary 11. A typical assumption is that communication is via *private channels*. That is, the faulty nodes' behaviour at time t is a function of all communication from correct nodes to faulty nodes during the interval $[0, t]$, the inputs, and the consensus algorithm only.

We can now obtain pulse synchronisation algorithms that are efficient both with respect to stabilisation time and communication. For example, we can make use of the randomised consensus algorithm by King and Saia [22].

Theorem 6 ([22] and Lemma 30)**.** *Suppose communication is via private channels. There is a family of randomised synchronous consensus routines that satisfy the following properties:*

- *the algorithm satisfies agreement with constant probability,*
- *the algorithm satisfies validity,*
- *the algorithm terminates in $R(f) \in \operatorname{polylog} f$ rounds,*
- *the number of bits broadcasted by each node in each round is $M(f) \in \operatorname{polylog} f$,*
- *and $n(f) > (3 + \varepsilon)f$ for a constant $\varepsilon > 0$ that can be freely chosen upfront.*

We point out that the algorithm by King and Saia actually satisfies stronger bounds on the total number of bits *sent* by each node than what is implied by our statement.

Corollary 2. *Suppose we have private channels. For any $f \geq 0$, constant $\varepsilon > 0$, and $n > (3+\varepsilon)f$, there exists a randomised f-resilient $\Theta(\operatorname{polylog} f)$-pulser over n nodes with skew $2d$ that stabilises in* $\operatorname{polylog} f$ *time w.h.p. and has nodes broadcast* $\operatorname{polylog} f$ *bits per time unit.*

Note that it is trivial to boost the probability for stabilisation by repetition, as the algorithm must stabilise in $\operatorname{polylog} f$ time regardless of the initial system state. However, in case of a uniform (or slowly growing) running time as function of f, it is useful to apply concentration bounds to show a larger probability of stabilisation. Concretely, the algorithm by Feldman and Micali offers constant expected running time, regardless of f; this translates to constant probability of success for an $O(1)$-round algorithm in our setting.

Theorem 7 ([19] and Lemma 30)**.** *Suppose that communication is via private channels. There exists a family of randomised synchronous consensus routines that satisfy the following properties:*

- *the algorithm satisfies agreement with constant probability,*
- *the algorithm satisfies validity,*
- *the algorithm terminates in $R(f) \in O(1)$ rounds,*
- *the total number of bits sent by each node is* $\operatorname{poly} f$*,*
- *and $n(f) = 3f + 1$.*

Employing this consensus routine, every $O(1)$ time units there is a constant probability that the next level of recursion stabilises. Applying Chernoff's bound over the (at most) $\log n$ recursive levels of stabilisation, this yields stabilisation in $O(\log n)$ time w.h.p.

Corollary 3. *Suppose we have private channels. For any $f \geq 0$ and $n > 3f$, there exists a randomised f-resilient $\Theta(\log f)$-pulser over n nodes with skew $2d$ that stabilises in $O(\log f)$ time w.h.p. and has nodes broadcast* $\operatorname{polylog} n$ *bits per time unit.*

Acknowledgements

We thank Danny Dolev for numerous discussions and ideas on the pulse synchronisation problem and detailed comments on the manuscript. We also thank Janne H. Korhonen, Jukka Suomela, and anonymous reviewers for helpful comments on the manuscript. Part of this work was done while JR was affiliated with Helsinki Institute for Information Technology HIIT, Department of Computer Science, Aalto University.

References

[1] Marcos Kawazoe Aguilera and Sam Toueg. Simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. *Information Processing Letters*, 71(3):155–158, 1999. doi:10.1016/S0020-0190(99)00100-3.

[2] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. 2nd Symposium on Principles of Distributed Computing (PODC 1983)*, pages 27–30. ACM, 1983. doi:10.1145/800221.806707.

[3] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Fast self-stabilizing Byzantine tolerant digital clock synchronization. In *Proc. 27th Annual ACM Symposium on Principles of Distributed Computing (PODC 2008)*, pages 385–394. ACM Press, 2008. doi:10.1145/1400751.1400802.

[4] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 410–415. IEEE, 1989. doi:10.1109/SFCS.1989.63511.

[5] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Bit optimal distributed consensus. In Ricardo Baeza-Yates and Udi Manber, editors, *Computer Science: Research and Applications*, pages 313–321. Springer US, 1992. doi:10.1007/978-1-4615-3422-8_27.

[6] Ariel Daliot, Danny Dolev, and Hanna Parnas. Self-stabilizing pulse synchronization inspired by biological pacemaker networks. In *Proc. 6th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2003)*, volume 2704 of *Lecture Notes in Computer Science*, pages 32–48. Springer, 2003. doi:10.1007/3-540-45032-7_3.

[7] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974. doi:10.1145/361179.361202.

[8] Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[9] Danny Dolev and Ezra N Hoch. Byzantine self-stabilizing pulse in a bounded-delay model. In *Proc. 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2007)*, volume 4838 of *Lecture Note in Computer Science*, pages 234–252. Springer, 2007. doi:10.1007/978-3-540-76627-8_19.

[10] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, 1985. doi:10.1145/2455.214112.

[11] Danny Dolev, Joseph Y. Halpern, and H.Raymond Strong. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Sciences*, 32(2): 230–250, 1986. doi:10.1016/0022-0000(86)90028-0.

[12] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3): 499–516, 1986. doi:10.1007/BF01783662.

[13] Danny Dolev, Matthias Függer, Christoph Lenzen, and Ulrich Schmid. Fault-tolerant algorithms for tick-generation in asynchronous logic: Robust pulse generation. *Journal of the ACM*, 61(5):30:1–30:74, 2014. doi:10.1007/978-3-642-24550-3_14. arXiv:1105.4780.

[14] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela, and Siert Wieringa. Synchronous counting and computational algorithm design. *Journal of Computer and System Sciences*, 82(2):310–332, 2016. doi: 10.1016/j.jcss.2015.09.002.

[15] Shlomi Dolev. *Self-Stabilization.* The MIT Press, Cambridge, MA, 2000.

[16] Shlomi Dolev and Jennifer L. Welch. Self-stabilizing clock synchronization in the presence of Byzantine faults. *Journal of the ACM*, 51(5):780–799, 2004. doi:10.1145/1017460.1017463.

[17] A. D. Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4(1):9–29, 1990. doi:10.1007/BF01783662.

[18] Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 148–161, 1988.

[19] Paul Feldman and Silvio Micali. An optimal probabilistic algorithm for synchronous Byzantine agreement. In *Proc. 16th International Colloquium on Automata, Languages and Programming (ICALP 1989)*, volume 372 of *Lecture Notes in Computer Science*, pages 341–378. Springer, 1989. doi:10.1007/BFb0035770.

[20] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982. doi:10.1016/0020-0190(82) 90033-3.

[21] Pankaj Khanchandani and Christoph Lenzen. Self-stabilizing Byzantine clock synchronization with optimal precision. In *Proc. 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2016)*, pages 213–230, 2016.

[22] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier. *Journal of the ACM*, 58(4): 1–24, 2011. doi:10.1145/1989727.1989732.

[23] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(I):52–78, 1985. doi:10.1145/2455.2457.

[24] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. doi: 10.1145/357172.357176.

[25] Christoph Lenzen and Joel Rybicki. Efficient counting with optimal resilience. In *Proc. 29th International Symposium on Distributed Computing (DISC 2015), Tokyo, Japan, October 7–9, 2015*, volume 9363 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2015. doi:10.1007/978-3-662-48653-5_2.

[26] Christoph Lenzen and Joel Rybicki. Near-optimal self-stabilising counting and firing squads. In *Proc. 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2016)*, pages 263–280, 2016. arXiv:1608.00214.

[27] Christoph Lenzen, Matthias Függer, Markus Hofstätter, and Ulrich Schmid. Efficient construction of global time in SoCs despite arbitrary faults. In *Proc. 16th Euromicro Conference on Digital System Design (DSD 2013)*, pages 142–151, 2013. doi:10.1109/DSD.2013.97.

[28] Christoph Lenzen, Joel Rybicki, and Jukka Suomela. Towards optimal synchronous counting. In *Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC 2015)*, pages 441–450. ACM, 2015. arXiv:1503.06702v1.

[29] Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2–3):190–204, 1984.

[30] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.

[31] Michael O. Rabin. Randomized Byzantine generals. In *Proc. 24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, pages 403–409. IEEE, 1983. doi: 10.1109/SFCS.1983.48.

[32] Michel Raynal. *Fault-tolerant agreement in synchronous message-passing systems.* Morgan & Claypool, 2010. doi:10.2200/S00294ED1V01Y201009DCT003.

[33] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3): 626–645, 1987. doi:10.1145/28869.28876.

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