A

Fault-tolerant Algorithms for Tick-Generation in Asynchronous Logic: Robust Pulse Generation

DANNY DOLEV, Hebrew University of Jerusalem, Israel
MATTHIAS FÜGGER and ULRICH SCHMID, Vienna University of Technology, Austria
CHRISTOPH LENZEN, Massachusetts Institute of Technology, USA

Today's hardware technology presents a new challenge in designing robust systems. Deep submicron VLSI technology introduces transient and permanent faults that were never considered in low-level system designs in the past. Still, robustness of that part of the system is crucial and needs to be guaranteed for any successful product. Distributed systems, on the other hand, have been dealing with similar issues for decades. However, neither the basic abstractions nor the complexity of contemporary fault-tolerant distributed algorithms match the peculiarities of hardware implementations.

This article is intended to be part of an attempt striving to bridge over this gap between theory and practice for the clock synchronization problem. Solving this task sufficiently well will allow to build an ultra-robust high-precision clocking system for hardware designs like systems-on-chips in critical applications. As our first building block, we describe and prove correct a novel distributed, Byzantine fault-tolerant, probabilistically self-stabilizing pulse synchronization protocol, called FATAL, that can be implemented using standard asynchronous digital logic: Correct FATAL nodes are guaranteed to generate pulses (i.e., unnumbered clock ticks) in a synchronized way, despite a certain fraction of nodes being faulty. FATAL uses randomization only during stabilization and, despite the strict limitations introduced by hardware designs, offers optimal resilience and smaller complexity than all existing protocols. Finally, we show how to leverage FATAL to efficiently generate synchronized, self-stabilizing, high-frequency clocks.

Categories and Subject Descriptors: F.2.m [**Analysis of Algorithms and Problem Complexity**]: Miscellanous; B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: clock synchronization, Byzantine faults, self-stabilization, multi-synchronous GALS, metastability, linear convergence time

ACM Reference Format:
Dolev, D., Függer, M., Lenzen, C., and Schmid, U., 2011. **F**ault-tolerant **A**lgorithms for **T**ick-Generation in **A**synchronous **L**ogic: Robust Pulse Generation. J. ACM V, N, Article A (January YYYY), 73 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

This work has been supported by the Swiss National Science Foundation (SNSF), by the Swiss Society of Friends of the Weizmann Institute of Science, by the Deutsche Forschungsgemeinschaft (DFG, reference number Le 3107/1-1), by the Austrian Science Foundation (FWF) project FATAL (P21694), by The Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11), by the ISG (Israeli Smart Grid) Consortium, administered by the office of the Chief Scientist of the Israeli ministry of Industry and Trade and Labor, and by grant 3/9778 of the Israeli Ministry of Science and Technology. Danny Dolev is incumbent of the Berthold Badler Chair.
Authors' addresses: D. Dolev, School of Engineering and Computer Science, The Hebrew University of Jerusalem, Givat Ram, 91904 Jerusalem, Israel; C. Lenzen, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), office G670, 32 Vassar Street, 02139 Cambridge, MA, USA; M. Függer, U. Schmid, Department of Computer Engineering, Vienna Institute of Technology, Treitlstrasse 3, 1040 Vienna, Austria.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION & RELATED WORK

Fault tolerance has been a focus of intense research both in theory and in practice. The two disciplines developed disjoint tools in order to increase the robustness of the envisioned systems: Theoretical studies, naturally, were discussing severe fault scenarios that have been considered too abstract to bear significance for real-world systems by practitioners. In contrast, engineers focused on typical settings that tend to be more benign, permitting to be treated efficiently. However, the ongoing rapid growth of system sizes results in ever-increasing varieties and frequencies of faults. Consequently, in the last decade, designers revisited some of the abstract models from theory, and such models slowly find their way into practical distributed applications.

This process is highly challenging, as the gap that needs to be bridged seems to have widened over time. To combine the best of both worlds, it is critical to (i) adapt and extend theoretical results to simultaneously address *all* relevant optimization criteria satisfactorily,[1] (ii) find suitable hierarchies of abstraction that allow for a seamless connection from high-level reasoning to the low-level building blocks (ultimately, basic logic gates and wires) of the system, while hiding away uncritical details on each level of abstraction; and (iii) redesign systems on all levels to respect the pivotal demands and fundamental limitations identified in the new framework. Naturally, this requires an unusually close interaction and thus communication between the two areas.

The present article is the result of this paradigm, applied to the problem of distributed clock generation for systems-on-chip (SoC). It emphasizes theory aspects, i.e., points (i) and (ii) of the above list. Regarding (iii), we refer the interested reader to Dolev et al. [2014]. We introduce a novel model and supporting techniques to express and analyze such algorithms. The model allows for a direct mapping of algorithms to hardware, yet provides sufficient abstraction to prove synchronization properties without any gap, i.e., violations of these properties can be tracked down to violations of the fault model expressed in terms of the very basic components of the system. Beyond that, our approach leads to a number of new results in both areas that we believe to be of independent interest.

Problem Motivation

With today's deep submicron technology running at GHz clock speeds [International Technology Roadmap for Semiconductors 2012], disseminating the high-speed clock throughout a *very large scale integrated* (VLSI) circuit, with negligible skew, is difficult and costly [Bhamidipati et al. 2002; Friedman 2001; Metra et al. 2004; Restle et al. 2001]. SoC are hence increasingly designed *globally asynchronous locally synchronous* (GALS) [Chapiro 1984], where different parts of the chip use different local clock signals. Two main types of clocking schemes for GALS systems exist, namely, (i) those where the local clock signals are unrelated, and (ii) multi-synchronous ones that provide a certain degree of synchrony between local clock signals [Semiat and Ginosar 2003; Teehan et al. 2007].

GALS systems clocked by type (i) permanently bear the risk of *metastable upsets* when conveying information from one clock domain to another. To explain the issue, consider a physical implementation of a bistable storage element, like a register cell, that can be accessed by read and write operations concurrently. It can be shown that two operations (like two writes with different values) occurring very close to each other can cause the storage cell to attain neither of its two stable states for an unbounded time [Marino 1981], and thereby, during an unbounded time afterwards, successive reads may return none of the stable states. Although the probability of a single upset

[1] Existing solutions require a long time to stabilize or exhibit a large communication complexity, and metastability is not addressed. These issues will be discussed later on.

is very small, one has to take into account that every bit of transmitted information across clock domains is a candidate for an upset. Elaborate synchronizers [Dike and Burton 1999; Kinniment et al. 2002; Portmann and Meng 1995] are the only means for achieving an acceptably low probability for metastable upsets here.

This problem can be circumvented in clocking schemes of type (ii): Common synchrony properties offered by multi-synchronous clocking systems are:

— *bounded skew*, i.e., bounded maximum time between the occurrence of any two matching clock transitions of any two local clock signals. Here, in classic clock synchronization, two clock transitions are matching iff they are both the k^{th}, $k \geq 1$, clock transition of a local clock.
— *bounded accuracy*, i.e., bounded minimum and maximum time between the occurrence of any two successive clock transitions of any local clock signal.

Type (ii) clocking schemes are particularly beneficial from a designer's point of view, since they combine the convenient local synchrony of a GALS system with a global time base across the whole chip. It has been shown by Polzer et al. [2009] that these properties indeed facilitate metastability-free high-speed communication across clock domains.

The decreasing structure sizes of deep submicron technology also resulted in an increased likelihood of chip components failing during operation: Reduced voltage swing and smaller critical charges make circuits more susceptible to ionized particle hits, crosstalk, and electromagnetic interference [Constantinescu 2003; Gadlage et al. 2006]. *Fault-tolerance* hence becomes an increasingly pressing issue in chip design. Unfortunately, faulty components may behave non-benign in many ways. They may perform signal transitions at arbitrary times and even convey inconsistent information to their successor components if their outgoing communication channels are affected by a failure. Well-known theory on fault-tolerant consensus and synchronization shows that this behavior is the key feature of unrestricted, i.e., *Byzantine* faults [Pease et al. 1980]. This forces to model faulty components as Byzantine if a high fault coverage is to be guaranteed.

The DARTS fault-tolerant clock generation approach [Függer et al. 2006; Függer et al. 2010] developed by some of the authors of this paper is a Byzantine fault-tolerant multi-synchronous clocking scheme. DARTS comprises a set of modules, each of which generates a local clock signal for a single clock domain. The DARTS modules (nodes) are synchronized to each other to within a few clock cycles. This is achieved by exchanging binary clock signals only, via a single wire in each direction between pairs of nodes. The basic idea behind DARTS is to employ a simple fault-tolerant distributed algorithm [Widder and Schmid 2009]—based on the consistent broadcasting primitive of Srikanth and Toueg [1987]—implemented in asynchronous digital logic. An important property of the DARTS clocking scheme is that it guarantees that no metastable upsets occur during fault-free executions. For executions with faults, metastable upsets cannot be ruled out: Since Byzantine faulty components are allowed to issue unrelated read and write accesses by definition, the same arguments as for clocking schemes of type (i) apply. However, Fuchs et al. [2009] show that by proper chip design, the probability of a Byzantine component leading to a metastable upset of DARTS can be made arbitrarily small, at a reasonable cost.[2]

Although both theoretical analysis and experimental evaluation revealed many attractive additional features of DARTS, like guaranteed startup, automatic adaption to

[2]Note that due to this cost, it is not desirable to rely on this mechanism for all communication; this would essentially mean to suffer the disadvantages of a type (i) GALS system despite the effort of synchronizing the different clock domains!

current operating conditions, etc., there is room for improvement. The most obvious drawback of DARTS is its inability to support late joining and restarting of nodes, and, more generally, its lack of self-stabilization properties. If, for some reasons, more than a third of the DARTS nodes ever become faulty, the system cannot be guaranteed to resume normal operation even if all failures cease. Even worse, simple transient faults such as radiation- or crosstalk-induced additional (or omitted) clock ticks accumulate over time to arbitrarily large skews in an otherwise benign execution.

Byzantine-tolerant *self-stabilization*, on the other hand, is the major strength of a number of protocols [Ben-Or et al. 2008; Daliot et al. 2003; Daliot and Dolev 2006; Dolev and Hoch 2007; Dolev and Welch 2004; Hoch et al. 2006; Malekpour 2006] primarily devised for distributed systems. These works reveal an interplay between the tasks of *pulse synchronization*—where the purpose is to generate well-separated anonymous pulses that are synchronized at all correct nodes—consensus, and clock synchronization. One can use consensus to agree on clocks and use clocks to dictate phases to run consensus. Pulse synchronization enables consensus on invocation of pulses within a small real-time window, where pulses are spaced in time, and between two consecutive pulses one can run consensus. Solving each of these problems from an arbitrary initial state poses a major challenge.

In light of these relations, the above papers on self-stabilizing pulse synchronization are of particular interest in the our context. Beyond optimal resilience (i.e., $\lceil n/3 \rceil - 1$, c.f. Pease et al. [1980]), an attractive feature of many of these protocols is a small stabilization time of $\mathcal{O}(n)$ [Ben-Or et al. 2008; Daliot and Dolev 2006; Dolev and Hoch 2007; Hoch et al. 2006; Malekpour 2006], which is crucial for applications with stringent availability requirements. In particular, Ben-Or et al. [2008] synchronize clocks in expected constant time in a synchronous system. Given any pulse synchronization protocol stabilizing in a bounded-delay system in expected time T, this implies an expected $(T + \mathcal{O}(1))$-stabilizing clock synchronization protocol.

Nonetheless, it remains open whether (with respect to the number of nodes n) a sublinear convergence time can be achieved: While the classical consensus lower bound of $f + 1$ rounds for synchronous, deterministic algorithms in a system with $f < n/3$ faults [Fischer and Lynch 1982] proves that *exact* consensus on a clock value requires at least $f + 1 \in \Omega(n)$ deterministic rounds, one has to face the fact that only approximate agreement on the current time is achievable in a bounded-delay system anyway. However, no non-trivial time lower bounds on approximate deterministic synchronization or the exact problem with randomization are currently known.

Note that existing synchronization algorithms, in particular those that do not rely on pulse synchronization, have deficiencies rendering them unsuitable in our context. For example, they have exponential convergence time [Dolev and Welch 2004], require the relative drift of the nodes' local clocks to be very small [Daliot et al. 2003; Dolev and Hoch 2007; Malekpour 2006],[3] are not applicable for $f > 1$ [Malekpour 2006] (cf. [Malekpour 2009]) or make use of linear-sized messages [Daliot and Dolev 2006; Dolev and Hoch 2007]. Furthermore, standard models used by the distributed systems community do not account for metastability, resulting in the same to be true for the existing solutions.

All previous pulse synchronization algorithms that stabilize in time $\mathcal{O}(n)$ make use of up to $\Omega(n)$ concurrently running consensus instances. The underlying idea is that a correct node can easily initiate a simulated synchronous execution of a consensus protocol to establish a common perception of some property of the system state among

[3]Note that it is too costly and space consuming to equip each node with a quartz oscillator. Simple digital oscillators, like inverters with feedback, in turn exhibit drifts of at least several percent, which heavily vary with operating conditions.

the nodes; because consensus is employed, misinformation by faulty nodes can be controlled sufficiently well to ensure eventual stabilization. The disadvantage is a large complexity, in terms of the local computations performed by each node, but, more importantly, also in terms of the required communication bandwidth of $\Omega(n)$ between each pair of nodes. Our key technical contribution is a way to achieve the same goal without consensus, which enables to reduce the required bandwidth to $\mathcal{O}(1)$ broadcasted bits per node in constant time.

Detailed Contributions

We describe and prove correct the novel FATAL pulse synchronization protocol, which facilitates a direct implementation in standard asynchronous digital logic.[4] An extended version of the protocol, termed FATAL$^+$, generates clock ticks (i.e., integer-labeled pulses) of improved skew and accuracy at a high frequency, as we discuss in Section 6. The protocol self-stabilizes within $\mathcal{O}(n)$ time with probability $1 - 2^{-(n-f)}$,[5] in the presence of $f < n/3$ Byzantine faulty nodes, and is metastability-free by construction after stabilization in failure-free runs (Theorem 6.2). While executing the protocol, non-faulty nodes broadcast a constant number of bits in constant time. In terms of distributed message complexity, this implies that stabilization is achieved after broadcasting $\mathcal{O}(n)$ messages of size $\mathcal{O}(1)$, improving by factor $\Omega(n \log n)$ on the number of bits transmitted by previous algorithms.[6] The protocol can sustain arbitrary relative clock drifts, which is crucial if the local clock sources are simple ring oscillators (uncompensated ring oscillators suffer from frequency variations of up to 20% [Sundaresan et al. 2006]), Lemma 3.4.

If the number of faults is not overwhelming, i.e., a majority of at least $n-f$ nodes continues to execute the protocol in an orderly fashion, recovering nodes and late joiners (re)synchronize in constant time (Theorem 4.17). This property is highly desirable in practical systems, in particular in combination with Byzantine fault-tolerance: Even if nodes randomly experience transient faults on a regular basis, quick recovery ensures that the mean time until failure of the system as a whole is substantially increased. All this is achieved against a powerful adversary that, at time t, knows the whole history of the system up to time t and does not need to choose the set of faulty nodes in advance (Corollary 5.5). For more benign settings—meaning that essentially the system is oblivious to the point in time when a majority of components becomes non-faulty—the stabilization time is constant in expectation (Corollary 5.6). Apart from bounded drifts and communication delays, our solution only requires that receivers can distinguish senders when receiving a message (i.e., it can identify the channel on which the incoming message is received, and there is a unique sender that uses this channel), which is a property that arises naturally in hardware designs.

We finally show how the pulse synchronization protocol can be extended to the FATAL+ clock synchronization protocol that computes bounded logical high-frequency clocks with small skew (Section 6). FATAL cannot generate pulses at very high frequency, yet we desire a high frequency of clock ticks. This is achieved by combining a simple non-self-stabilizing Byzantine tolerant pulse synchronization algorithm with FATAL. Adding a high-frequency oscillator to derive even faster clocks from FATAL$^+$

[4] In Dolev et al. [2014], we describe how the pulse synchronization protocol can be implemented using asynchronous digital logic.

[5] Note that the algorithm by Ben-Or et al. [2008] achieving an expected constant stabilization time in a synchronous model needs to run for $\Omega(n)$ rounds to ensure the same probability of stabilization.

[6] We remark that Malekpour [2006; Malekpour [2009], which conceptually achieves the same complexity, considers a much simpler model. In particular, individual clocks do not drift apart and the protocol can sustain a single Byzantine fault only.

is straightforward. Again, in the absence of faulty components the extended protocol provably does not suffer from metastability once stabilized. During stabilization, the fact that nodes merely undergo a constant number of state transitions in constant time ensures a very small probability of metastable upsets.

Organization of the Article

The remainder of this work is structured as follows. In Section 2, we introduce the formal framework in which our algorithm operates, and precisely state the problem to be solved. Essentially, we utilize asynchronous state machines that communicate via bounded-delay FIFO channels and have access to bounded clocks of bounded relative drift. However, many details require careful attention, such as appropriately modeling metastability and randomization. We proceed by presenting the FATAL pulse synchronization algorithm in Section 3. As pseudo-code is not well-suited to represent the logical flow of the algorithm, we derive a condensed graphical representation that facilitates grasping the main concepts, yet allows for a full specification of the algorithm. This approach is complemented by a textual description that focuses on central ideas and typical executions of the algorithm. The section concludes with a listing of a number of constraints the parameters of the algorithm must satisfy and a solution to the respective system of inequalities.

Subsequently, we prove in Section 4 that the given algorithm indeed is a self-stabilizing pulse synchronization algorithm as formulated in Section 2. Apart from showing correctness, the statements are organized in a way attempting to highlight the crucial properties of the algorithm. Moreover, the proofs clarify the details omitted from the verbal description of the algorithm in Section 3, and connect the constraints on the algorithm's parameters to its structural properties. The more involved proofs are prefaced by outlines summarizing their key arguments. Section 5 provides additional results that would have complicated or impeded the presentation of the core proofs in Section 4.

We next present how to extend the pulse synchronization algorithm to the FATAL+ clock synchronization protocol in Section 6. While we stress that this is neither the sole possibility to derive synchronized high-frequency clocks from pulse synchronization nor its only application, we believe that this example convincingly confirms the relevance of self-stabilizing pulse synchronization for self-stabilizing clock synchronization. Finally, Section 7 reviews the algorithm in terms of asymptotic complexity, which also includes implementation complexity, and concludes the article with an outlook on future work.

2. MODEL

In this section, we introduce our system model. Our formal framework will be tied to the peculiarities of hardware designs, which consist of modules that *continuously*[7] compute their output signals based on their input signals.

Signals

Following Függer [2010]; Függer and Schmid [2012], we define (the trace of) a *signal* to be a timed event trace over a finite alphabet $\mathbb{S}$ of possible signal states: Formally, signal $\sigma \subseteq \mathbb{S} \times \mathbb{R}_0^+$. All times and time intervals refer to a global *reference time* taken from $\mathbb{R}_0^+$, that is, signals reflect the system's state from time 0 on. The elements of σ are called *events*, and for each event (s,t) we call s the *state of event* (s,t) and t the *time of event* (s,t). In general, a signal σ is required to fulfill the following conditions: (i) for

[7]In sharp contrast to classic distributed computing models, there is no computationally complex discrete zero-time state-transition here.

each time interval $[t^-, t^+] \subseteq \mathbb{R}_0^+$ of finite length, the number of events in σ with times within $[t^-, t^+]$ is finite, (ii) from $(s,t) \in \sigma$ and $(s',t) \in \sigma$ follows that $s = s'$, and (iii) there exists an event at time 0 in σ.

Note that our definition allows for events (s,t) and $(s,t') \in \sigma$, where $t < t'$, without having an event $(s',t'') \in \sigma$ with $s' \neq s$ and $t < t'' < t'$. In this case, we call event (s,t') *idempotent*. Two signals σ and σ' are *equivalent*, iff they differ in idempotent events only. We identify all signals of an equivalence class, as they describe the same physical signal. Each equivalence class $[\sigma]$ of signals contains a unique signal σ_0 having no idempotent events. We say that *signal σ switches to s* at time t iff event $(s,t) \in \sigma_0$.

The *state of signal* σ at time $t \in \mathbb{R}_0^+$, denoted by $\sigma(t)$, is given by the state of the event with the maximum time not greater than t.[8] Because of (i), (ii) and (iii), $\sigma(t)$ is well defined for each time $t \in \mathbb{R}_0^+$. Note that σ's state function in fact depends on $[\sigma]$ only, i.e., we may add or remove idempotent events at will without changing the state function.

Distributed System

On the topmost level of abstraction, we see the system as a set of $V = \{1, \dots, n\}$ physically remote *nodes* and communication *channels* between all nodes, over which nodes can broadcast their states. In the context of a VLSI circuit, "physically remote" actually refers to quite small distances (centimeters or even less). However, at gigahertz frequencies, a local state transition will not be observed remotely within a time that is negligible compared to clock speeds. We stress this point, since it is crucial that different clocks (and their attached logic) are not placed too close to each other, as otherwise they might fail due to the same physical error. This would render it pointless to devise a system that is resilient to a certain fraction of the nodes failing.

Each node i comprises a number of *input ports*, namely $S_{i,j}$ for each node j, an *output port* S_i, and a set of *local ports*, introduced later on. An *execution* of the distributed system assigns to each port of each node a signal. For convenience of notation, for any port p, we refer to the signal assigned to port p in execution $\mathcal{E}$ simply by signal p in execution $\mathcal{E}$, or just signal p if the execution is clear from the context. We say that *node i is in state s* at time t iff $S_i(t) = s$. We further say that *node i switches to state s* at time t iff signal S_i switches to s at time t.

Nodes exchange their states via the channels between them: for each pair of nodes i, j, output port S_i is connected to input port $S_{j,i}$ by a FIFO channel from i to j. Note that this includes a channel from i to i itself. Intuitively, S_i being connected to $S_{j,i}$ by a (non-faulty) channel means that $S_{j,i}(\cdot)$ should mimic $S_i(\cdot)$, however, with a slight delay accounting for the time it takes the channel to propagate events. In contrast to an asynchronous system, this delay is bounded by the *maximum delay* $d > 0$.[9]

Formally we define: The *channel* from node i to j is said to be *correct* during $[t^-, t^+]$ iff there exists a function $\tau_{i,j} : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, called the channel's *delay function*, such that: (i) $\tau_{i,j}$ is continuous and strictly increasing, (ii) $\forall t \in [\max(t^-, \tau_{i,j}(0)), t^+] : 0 < t - \tau_{i,j}^{-1}(t) < d$, and (iii) for each $t \in [\max(t^-, \tau_{i,j}(0)), t^+]$, $(s,t) \in S_{j,i} \Leftrightarrow (s, \tau_{i,j}^{-1}(t)) \in S_i$, and for each $t \in [t^-, \tau_{i,j}(0))$, $(s,t) \in S_{j,i} \Rightarrow s = S_i(0)$. Note that because of (i), $\tau_{i,j}^{-1}$ exists in the domain $[\tau_{i,j}(0), \infty)$, and thus (ii) and (iii) are well defined. We say that node i *observes node j in state s* at time t if $S_{i,j}(t) = s$.

[8] To facilitate intuition, we here slightly abuse notation, as this way σ denotes both a function of time and the signal (trace), which is a subset of $\mathbb{S} \times \mathbb{R}_0^+$. Whenever referring to σ, we will talk of the signal, not the state function.

[9] With respect to $\mathcal{O}$-notation, we normalize $d \in \mathcal{O}(1)$, as all time bounds simply depend linearly on d.

Clocks and Timeouts

Nodes are never aware of the current reference time and we also do not require the reference time to resemble Newtonian "real" time. Rather we allow for physical clocks that run arbitrarily fast or slow,[10] as long as their speeds are close to each other in comparison. One may hence think of the reference time as progressing at the speed of the currently slowest correct clock. In this framework, nodes essentially make use of clocks with bounded drift.

Formally, clock rates are within $[1, \vartheta]$ (with respect to reference time), where $\vartheta > 1$ is constant and $\vartheta - 1$ is the *(maximum) clock drift*. A *clock* C is a continuous, strictly increasing function $C : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ mapping reference time to some local time. Clock C is said to be *correct* during $[t^-, t^+] \subseteq \mathbb{R}_0^+$ iff we have for any $t, t' \in [t^-, t^+]$, $t < t'$, that $t' - t \leq C(t') - C(t) \leq \vartheta(t' - t)$. Each node comprises a set of clocks assigned to it, which allow the node to estimate the progress of reference time. However, clearly unbounded, continuous valued clocks cannot be implemented by realistic hardware components. Nodes having access to such devices thus would be an unrealistically strong assumption. Also, we remark that in our context the actual value of a local clock is meaningless for stabilization from arbitrary initial states.

To account for this, a node may not directly access the value of its clocks, but rather only has access to so-called *timeout ports* of watchdog timers. Intuitively, a timeout is a counter that is started upon switching to a certain state and expires after a given period of time has passed on an associated local clock. However, our definition needs to account for the fact that an implementation cannot realize an instant reset of a timeout, as well as make sure that, in order to guarantee self-stabilization, a timeout will eventually expire even if it is in an inconsistent state when becoming correct.[11]

Formally, a *timeout* is a triple (T, s, C), where $T \in \mathbb{R}^+$ is a duration, $s \in \mathbb{S}$ is a state, and C is some local clock (each node may have several), say of node i. Each timeout (T, s, C) has a corresponding timeout port $\mathrm{Time}_{T,s,C}$, being part of node i's local input ports. Signal $\mathrm{Time}_{T,s,C}$ is Boolean, that is, its possible states are from the set $\{0, 1\}$. We say that timeout (T, s, C) is *correct* during $[t^-, t^+] \subseteq \mathbb{R}_0^+$ iff clock C is correct during $[t^-, t^+]$ and the following holds:

(1) For each time $t_s \in [t^-, t^+]$ when node i (i.e., its state function S_i) switches to state s, there is a time $t \in [t_s, \tau_{i,i}(t_s)]$ such that (T, s, C) is *reset*, i.e., $(0, t) \in \mathrm{Time}_{T,s,C}$.
(2) If $\mathrm{Time}_{T,s,C}(t^-) = 0$ and $t \leq t^+$ satisfies that $C(t) - C(t^-) = T$, there is a minimal time $t' \in [t^-, t]$ when the timeout is either reset or it *expires*, i.e., $\mathrm{Time}_{T,s,C}$ switches to 1.
(3) If the timeout is reset at time $t_s \in [t^-, t^+]$ and there is a time $t \in (t_s, t^+]$ such that (i) $C(t) - C(t_s) = T$ and (ii) the timeout is not reset at any time from $(t_s, t]$, then the timeout expires at time t.
(4) The timeout neither expires nor is it reset at any time not specified above.

We say that timeout (T, s, C) *is expired* at time t iff $\mathrm{Time}_{T,s,C}(t) = 1$. For notational convenience, we will omit the clock C and simply write (T, s) for both the timeout and its signal.

A *randomized timeout* is a triple $(\mathcal{D}, s, C)$. Here, $\mathcal{D}$ is a discrete random distribution on a finite interval $I \subset \mathbb{R}_0^+$. Further $s \in \mathbb{S}$ is a state, and C is a clock. Its corresponding timeout port $\mathrm{Time}_{\mathcal{D},s,C}$ behaves very similar to the one of an ordinary timeout, except that whenever it is reset, the local time that passes until it expires next—provided

[10] Note that the formal definition excludes trivial solutions by requiring clocks' progress to be in a linear envelope of the reference time, see below.
[11] This is a fundamental requirement, as otherwise the initial state of the system could be deadlocked.

that it is not reset again before that happens—follows the distribution $\mathcal{D}$. Formally, $(\mathcal{D}, s, C)$ is correct during $[t^-, t^+] \subseteq \mathbb{R}_0^+$, if C is correct during $[t^-, t^+]$ and the following holds:

(1) For each time $t_s \in [t^-, t^+]$ when node i switches to state s, there is a time $t \in [t_s, \tau_{i,i}(t_s)]$ such that $(\mathcal{D}, s, C)$ is *reset*, i.e., $(0, t) \in \mathrm{Time}_{\mathcal{D},s,C}$.
(2) Let $D \subset I$ be the (finite) set of points so that for $x \in D$ it holds that $P[\mathcal{D} = x] \neq 0$. If $\mathrm{Time}_{T,s,C}(t^-) = 0$ and $t \leq t^+$ satisfies that $C(t) - C(t^-) = \max D$, there is a minimal time $t' \in [t^-, t]$ when the timeout is either reset or it *expires*, i.e., $\mathrm{Time}_{T,s,C}$ switches to 1.
(3) If the timeout is reset at time $t_s \in [t^-, t^+]$ and there is a time $t \in (t_s, t^+]$ such that (i) $C(t) - C(t_s) = x \in D$ and (ii) the timeout is not reset at any time from $(t_s, t]$, then $(1, t) \in \mathrm{Time}_{\mathcal{D},s,C}$ with probability

$$P[\mathcal{D} = x | \mathcal{D} \geq x] = \frac{P[\mathcal{D} = x]}{P[\mathcal{D} \geq x]} ,$$

independently of any other properties of the execution. In other words, whether $(1, t) \in \mathrm{Time}_{\mathcal{D},s,C}$ is determined at time t by an independent flip of a (biased) coin, and in case $(1, t) \notin \mathrm{Time}_{\mathcal{D},s,C}$, no additional information on when the timeout is going to expire is revealed.
(4) The timeout neither expires nor is it reset at any time not specified above.

We will apply the same notational conventions to randomized timeouts as we do for regular timeouts. Moreover, for the sake of a straightforward presentation, our algorithm will assume an idealized randomized timeout with a perfectly uniform distribution $\mathcal{D}$, which can be obtained as the limit of a sequence of discrete distributions. It is not hard to see that the analysis can be performed equally well with an equidistant discretization of the perfect uniform distribution of step width $\mathcal{O}(d)$ without affecting the asymptotic of the timeout bounds stated in Condition 3.3; the effect on the timeouts can be made arbitrarily small by sending the discretization parameter to 0.

The reason why we require that the probability of $(1, t) \in \mathrm{Time}_{\mathcal{D},s,C}$ must be *independent* of other properties of the execution is that we must avoid that the portion of a node's behavior that is randomized can be predicted.[12]

We remark that the above definition does not map the random distribution $\mathcal{D}$ and the clock C to the distribution of the next time $t \in [t_0, t^+)$ satisfying that $(1, t) \in \mathrm{Time}_{\mathcal{D},s,C}$, for two reasons. Firstly, re-entering state s might reset the timeout before a coin flip results in $(1, t) \in \mathrm{Time}_{\mathcal{D},s,C}$ for some t. Secondly, the definition permits that $(1, t), (1, t') \in \mathrm{Time}_{\mathcal{D},s,C}$ for $t' > t$ without the timeout being reset between t and t'. However, in the latter case the event $(1, t')$ will be idempotent and of no significance for the state of the timeout port $\mathrm{Time}_{\mathcal{D},s,C}$. Thus, for two subsequent times $t_0, t_0' \in [t^-, t^+]$ when the node switches to s, we get the meaningful statement that for any $t \in [t_0, t_0')$,

$$P[\mathrm{Time}_{\mathcal{D},s,C} \text{ switches to } 1 \text{ at time } t] = P[\mathcal{D} = C(t) - C(t_0)] .$$

We remark that these definitions allow for different timeouts to be driven by the same clock, implying that an adversary may derive some information on the state of a randomized timeout before it expires from the node's behavior, even in case it cannot directly access the values of the clock driving the timeout. However, in practice it might be very difficult to guarantee that the behavior of a dedicated clock that drives

[12] This is a non-trivial property. In particular, by drawing from a known random distribution of the expiration time, nodes could just determine, at time t_0, at which local clock value the timeout shall expire next. This would, however, essentially reveal when the timeout will expire prematurely, greatly reducing the power of randomization!

a randomized timeout is indeed independent of the execution of the algorithm. On the other hand, the more general definition imposes fewer restriction on the implementation and is thus preferrable.

Memory Flags

Besides timeout and randomized timeout ports, another kind of node i's local ports are *memory flags*. For each state $s \in \mathbb{S}$ and each node $j \in V$, $\mathrm{Mem}_{i,j,s}$ is a local port of node i. It is used to memorize whether node i has observed node j in state s since the last reset of the flag. We say that node i *memorizes node j in state s* at time t if $\mathrm{Mem}_{i,j,s}(t) = 1$. Formally, we require that signal $\mathrm{Mem}_{i,j,s}$ switches to 1 at time t iff node i observes node j in state s at time t and $\mathrm{Mem}_{i,j,s}$ is not already in state 1. The times t when $\mathrm{Mem}_{i,j,s}$ is *reset*, i.e., $(0,t) \in \mathrm{Mem}_{i,j,s}$, are specified by node i's state machine, which is introduced next.

State Machine

It remains to specify how nodes switch states and when they reset memory flags. We do this by means of state machines that may attain states from the finite alphabet $\mathbb{S}$. A node's state machine is specified by (i) the set $\mathbb{S}$, (ii) a function tr, called the *transition function*, from $\mathcal{T} \subseteq \mathbb{S}^2$ to the set of Boolean predicates on the alphabet consisting of expressions "$p = s$" (used for expressing guards), where p is from the node's input and local ports and s is from the set of possible states of signal p, and (iii) a function re, called the *reset function*, from $\mathcal{T}$ to the power set of the node's memory flags.

Intuitively, the transition function specifies the conditions (guards) under which a node switches states, and the reset function determines which memory flags to reset upon the state change. Formally, let P be a predicate on node i's input and local ports. We define *P holds at time t* by structural induction: If P is equal to $p = s$, where p is one of node i's input and local ports and s is one of the states signal p can obtain, then *P holds at time t* iff $p(t) = s$. Otherwise, if P is of the form $\neg P_1$, $P_1 \wedge P_2$, or $P_1 \vee P_2$, we define *P holds at time t* in the straightforward manner.

We say node i *follows its state machine during* $[t^-, t^+]$ iff the following holds: Assume node i observes itself in state $s \in \mathbb{S}$ at time $t \in [t^-, t^+]$, i.e., $S_{i,i}(t) = s$. Then, for each $(s, s') \in \mathcal{T}$, both:

(1) Node i switches to state s' at time t iff $tr(s, s')$ holds at time t and i is not already in state s'.[13] (In case more than one guard $tr(s, s')$ is true at the same time, we assume that an arbitrary tie-breaking ordering exists among the transition guards that specifies to which state to switch.)
(2) Node i resets memory flag m at some time in the interval $[t, \tau_{i,i}(t)]$ iff $m \in re(s, s')$ and i switches from state s to state s' at time t. This correspondence is one-to-one.

A node is defined to be *non-faulty* during $[t^-, t^+]$ iff during $[t^-, t^+]$ all its timeouts and randomized timeouts are correct and it follows its state machine. If it employs multiple state machines (see below), it needs to follow all of them.

In contrast, a faulty node may change states arbitrarily. Note that while a faulty node may be forced to send consistent output state signals to all other nodes if its channels remain correct, there is no way to guarantee that this still holds true if channels are faulty.[14]

[13] Recall that for a short period a node may still observe itself in state s albeit already having switched to s'.
[14] A single physical fault may cause this behavior, as at some point a node's output port must be connected to remote nodes' input ports. Even if one places bifurcations at different physical locations striving to mitigate this effect, if the voltage at the output port drops below specifications, the values of corresponding input channels may deviate in unpredictable ways.

Metastability

In our discrete system model, the effect of metastability is captured by the lacking capability of state machines to instantaneously take on new states: Node i decides on state transitions based on the delayed status of port $S_{i,i}$ instead of its "true" current state S_i. This non-zero delay from S_i to $S_{i,i}$ bears the potential for metastability, as a successful state transition can only be guaranteed if after a transition guard from some state s to some state s' becomes true, all other transition guards from s to $s'' \neq s'$ remain false during this delay at least.

This is exemplified in the following scenario: Assume node i is in state s at some time t. However, since it switched to s only very recently, it still observes itself in state $s' \neq s$ at time t via $S_{i,i}$. Given that there is a transition (s', s'') in $\mathcal{T}$, $s'' \neq s$, whose condition is fulfilled at time t, it will switch to state s'' at time t (although state s has not even stabilized yet). That is, due to the discrepancy between $S_{i,i}$ and S_i, node i switches from state s to state s'' at time t even if (s, s'') is not in $\mathcal{T}$ at all.[15]

In a physical chip design, these invalid changes of state might even result in inconsistent operations on the local memory, up to the point where it cannot be properly described in terms of $\mathbb{S}$, and thus in terms of our discrete model, anymore. Even worse, the state of i is part of the local memory and the node's state signal may attain an undefined value that is propagated to other nodes and their memory. While avoiding the latter is the task of the input ports of a non-faulty node, our goal is to prevent this erroneous behavior in situations where input ports attain legitimate values only.

Therefore, we define node i to be *metastability-free* in an execution, if the situation described above does not occur.

Definition 2.1 (*Metastability-Freedom*). Node $i \in V$ is called *metastability-free during* $[t^-, t^+]$ *(in execution* $\mathcal{E}$*)*, iff for each time $t \in [t^-, t^+]$ when i switches to some state $s \in \mathbb{S}$ in execution $\mathcal{E}$, it holds that $\tau_{i,i}(t) < t'$, where t' is the infimum of all times in $(t, t^+]$ when i switches to some state $s' \in \mathbb{S}$ in $\mathcal{E}$.

Multiple State Machines

In some situations the previous definitions are too stringent, as there might be different "components" of a node's state machine that act concurrently and independently, mostly relying on signals from disjoint input ports or orthogonal components of a signal. We model this by permitting that nodes run several state machines in parallel. All these state machines share the input and local ports of the respective node and are required to have disjoint state spaces. If node i runs state machines $M_1, \dots, M_k$, node i's output signal is the product of the output signals of the individual machines. Formally we define: Each of the state machines M_j, $1 \leq j \leq k$, has an additional own output port s_j. The state of node i's output port S_i at any time t is given by $S_i(t) := (s_1(t), \dots, s_k(t))$, where the signals of ports $s_1, \dots, s_k$ are defined analogously to the signals of the output ports of state machines in the single state machine case. Note that by this definition, the only (local) means for node i's state machines to interact with each other is by reading the delayed state signal $S_{i,i}$.

We say that *node i's state machine M_j is in state s at time t* iff $s_j(t) = s$, where $S_i(t) = (s_1(t), \dots, s_k(t))$, and that *node i's state machine M_j switches to state s at time t* iff signal s_j switches to s at time t. Since the state spaces of the machines M_j are disjoint, we will omit the phrase "state machine M_j" from the notation, i.e., we write "node i is in state s" or "node i switched to state s", respectively.

[15] Note that while the "internal" delay $\tau_{i,i}(t) - t$ can be made quite small, it cannot be reduced to zero if the model is meant to reflect physical implementations.

Recall that the various state machines of node i are as loosely coupled as remote nodes, namely via the delayed status signal on channel $S_{i,i}$ only. Therefore, it makes sense to consider them independently also when it comes to metastability.

Definition 2.2 (*Metastability-Freedom – Multiple State Machines*). We denote state machine M of node $i \in V$ as being *metastability-free during* $[t^-, t^+]$ *(in execution* $\mathcal{E}$*)*, iff for each time $t \in [t^-, t^+]$ when M switches from some state $s \in \mathbb{S}$ to another state $s' \in \mathbb{S}$ in execution $\mathcal{E}$, it holds that $\tau_{i,i}(t) < t'$, where t' is the infimum of all times in $(t, t^+]$ when M switches to some state $s'' \in \mathbb{S}$ in $\mathcal{E}$.

Note that by this definition the different state machines may switch states concurrently without suffering from metastability.[16] It is even possible that some state machine suffers metastability, while another is not affected by this at all.[17]

Problem Statement

The purpose of the pulse synchronization protocol is that nodes generate synchronized, well-separated pulses by switching to a distinguished state *accept*. Self-stabilization requires that it starts to do so within a bounded time, for any possible initial state. However, as our protocol makes use of randomization, there are executions where this does not happen at all; instead, we will show that the protocol stabilizes with probability one in finite time.

In a nutshell, our algorithm tolerates an adversary type that is commonly referred to as a *strong adversary*. An execution can be seen as a sequence of random experiments, namely the non-faulty nodes' coin flips, where the non-faulty components behave according to their specifications and all other aspects of the execution (faulty nodes' behavior, the rate of non-faulty clocks within $[1, \vartheta]$, etc.) are under the full control of the adversary. Our goal is to achieve stabilization quickly with a large probability (over the non-faulty nodes' coin flips) for any possible strategy of the adversary.

To give a precise meaning to this statement, we define the following probability spaces.

Definition 2.3 (*Adversarial Spaces*). Denote for $i \in V$ by $\mathcal{C}_i = (C_{i,1}, \dots, C_{i,c_i})$ the tuple of clocks of node i. An *adversarial space* is a probabilistic space that is defined by subsets of nodes $W \subseteq V$ and channels $E \subseteq V^2$, a time interval $[t^-, t^+]$, a protocol $\mathcal{P}$ (nodes' ports, state machines, etc.) as previously defined, the tuple of all clocks $(\mathcal{C}_1, \dots, \mathcal{C}_n)$, a function Θ assigning each $(i,j) \in V^2$ a delay $\tau_{i,j} : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, an initial state $\mathcal{E}_0$ of all ports, and an *adversarial function* $\mathcal{A}$. Here $\mathcal{A}$ is a function that maps a partial execution $\mathcal{E}|_{[0,t]}$ until time t (i.e., all ports' values until time t), W, E, $[t^-, t^+]$, $\mathcal{P}$, $\mathcal{C}$, and Θ to the states of all faulty ports during the time interval (t, ∞).

The adversarial space $\mathcal{AS}(W, E, [t^-, t^+], \mathcal{P}, \mathcal{C}, \Theta, \mathcal{E}_0, \mathcal{A})$ is now defined on the set of all executions $\mathcal{E}$ satisfying that (i) the initial state of all ports is given by $\mathcal{E}|_{[0,0]} = \mathcal{E}_0$, (ii) for all $i \in V$ and $k \in \{1, \dots, c_i\} : C^{\mathcal{E}}_{i,k} = C_{i,k}$, (iii) for all $(i,j) \in V^2$, $\tau^{\mathcal{E}}_{i,j} = \tau_{i,j}$, (iv) nodes in W are non-faulty during $[t^-, t^+]$ with respect to the protocol $\mathcal{P}$, (v) all channels in E are correct during $[t^-, t^+]$, and (vi) given $\mathcal{E}|_{[0,t]}$ for any time t, $\mathcal{E}|_{(t,t']}$ is given by $\mathcal{A}$, where t' is the infimum of times greater than t when a non-faulty node switches

[16] However, care has to be taken when implementing the inter-node communication of the state components in a metastability-free manner.

[17] This is crucial for the algorithm we are going to present. For stabilization purposes, nodes comprise a state machine that is prone to metastability. However, the state machine generating pulses (i.e., having the state *accept*, cf. Definition 2.4) does not take its output signal into account once stabilization is achieved. Thus, the algorithm is metastability-free after stabilization in the sense that we guarantee a metastability-free signal indicating when pulses occur.

states. Thus, except for when randomized timeouts expire, $\mathcal{E}$ is fully predetermined by the parameters of $\mathcal{AS}$.[18] The probability measure on $\mathcal{AS}$ is induced by the random distributions of the randomized timeouts specified by $\mathcal{P}$.

To avoid confusion, observe that if the clock functions and delays do not follow the model constraints during $[t^-, t^+]$, the respective adversarial space is empty and thus of no concern. This, admittedly cumbersome, definition provides the means to formalize a notion of stabilization that accounts for worst-case drifts and delays and an adversary that knows the full state of the system up to the current time.

We are now in the position to formally state the pulse synchronization problem in our framework. Intuitively, the goal is that after transient faults cease, nodes should with probability one eventually start to issue well-separated, synchronized pulses by switching to a dedicated state *accept*. Thus, as the initial state of the system is arbitrary, specifying an algorithm[19] is equivalent to defining the state machines that run at each node, one of which has a state *accept*.

Definition 2.4 (*Self-Stabilizing Pulse Synchronization*). Given a set of nodes $W \subseteq V$ and a set $E \subseteq V \times V$ of channels, we say that protocol $\mathcal{P}$ is a (W, E)-*stabilizing pulse synchronization protocol with skew* Σ *and accuracy bounds* $T^- > \Sigma$ *and* T^+ *that stabilizes within time* T *with probability* p iff the following holds. Choose any time interval $[t^-, t^+] \supseteq [t^-, t^- + T + \Sigma]$ and any adversarial space $\mathcal{AS}(W, E, [t^-, t^+], \mathcal{P}, \cdot, \cdot, \cdot, \cdot)$ (i.e., $\mathcal{C}$, Θ, $\mathcal{E}_0$, and $\mathcal{A}$ are arbitrary). Then executions from $\mathcal{AS}$ satisfy with probability at least p that there exists a time $t_s \in [t^-, t^- + T]$ so that, denoting by $t_i(k)$ the time when node $i \in W$ switches to a distinguished state *accept* for the k^{th} time after t_s ($t_i(k) = \infty$ if no such time exists),

(i) $t_i(1) \in (t_s, t_s + \Sigma)$,
(ii) $|t_i(k) - t_j(k)| \leq \Sigma$ if $\max\{t_i(k), t_j(k)\} \leq t^+$,
(iii) $|t_i(k+1) - t_i(k)| \geq T^-$ if $t_i(k+1) \leq t^+$, and
(iv) $|t_i(k+1) - t_i(k)| \leq T^+$ if $t_i(k) + T^+ \leq t^+$.

Note that the fact that $\mathcal{A}$ is a deterministic function and, more generally, that we consider each space $\mathcal{AS}$ individually, is no restriction: As $\mathcal{P}$ succeeds for any adversarial space with probability at least p in achieving stabilization, the same holds true for randomized adversarial strategies $\mathcal{A}$ and worst-case drifts and delays.

3. THE FATAL PULSE SYNCHRONIZATION PROTOCOL

In this section, we present our self-stabilizing pulse generation algorithm. In order to be suitable for implementation in hardware, it needs to utilize very simple rules only. It is stated in terms of state machines as introduced in the previous section.

The overall idea of the pulse synchronization algorithm is as follows. In a stable state, the nodes will switch to the *accept* state in a well-synchronized fashion, then sleep for some time (by means of timeouts) and then initialize a fault-tolerant voting procedure to re-establish tight synchronization and generate the next pulse. The sleeping phase here makes sure that the voting signals that are considered for, say, the k^{th} pulse, are indeed for this pulse and not lingering signals from pulse $k-1$.

This strategy fails in case the nodes stop to follow the above scheme in a synchronized fashion. In order to recover this behavior from arbitrary states, the algorithm

[18] This follows by induction starting from the initial configuration $\mathcal{E}_0$. Using $\mathcal{A}$, we can always extend $\mathcal{E}$ to the next time when a correct node switches states, and when non-faulty nodes switch states is fully determined by the parameters of $\mathcal{AS}$ except for when randomized timeouts expire. Note that the induction reaches any finite time within a finite number of steps, as signals switch states finitely often in finite time.

[19] We use the terms "algorithm" and "protocol" interchangeably throughout this work.

needs to succeed in generating one pulse to "restart" the system in case the pulse generation deadlocks. To this end, nodes detecting that less than $n - f$ nodes are synchronized will switch to a *recover* state. Once all non-faulty nodes are in this state and a long timeout expired, a synchronized pulse is again generated by means of voting. However, it may also be the case that a subset of the nodes keeps generating pulses without ever detecting that system-wide synchronization has been lost. In this case, the recovering nodes need to join the ongoing pulse generation. Careful design of the algorithm avoids that these two mechanisms interfere. Ultimately, we rely on generation of a "randomized pulse" from a lower layer to establish a minimal amount of consistency among all non-faulty nodes' states. This facilitates a (sufficiently) consistent perception of the system state by all non-faulty nodes so that the two mechanisms can coexist without impeding each other's success. After stabilization, the lower layer is ignored by the pulse generation mechanism, therefore the algorithm provides deterministic guarantees during stable operation.

Since the ultimate goal of the pulse generation algorithm is to interact with an application layer (cf. Section 6), we introduce a possibility for a coupling with such a layer in the pulse generation algorithm itself. For each node i, we add a further port NEXT_i, which can be driven by node i's application layer. As for other state signals, its output raises flag $\text{Mem}_{i,\text{NEXT}}$, which for simplicity we call NEXT_i as well. This mechanism enables to use the slow pulses generated by the algorithm from this layer as "phase delimiters" that serve to stabilize the application layer algorithm, but avoid to disrupt the application layer's correct operation once stabilization is achieved. The latter is done by allowing the application layer to influence the time between two successively generated pulses within a range that does not prevent the pulse generation algorithm from stabilizing correctly.

Many application layers may not require this feedback mechanism. In Section 6, we give an example of an application layer that indeed does rely on the NEXT_i signals: We show how to couple a non-self-stabilizing clock synchronization routine with the pulse generation algorithm to make the former self-stabilizing. The feedback mechanism here serves a two-fold purpose. First, it avoids the need to directly force clock transitions of the clock synchronization layer on occurrence of a pulse whenever the clocks are running properly, which enables a tighter synchronization guarantee than provided by the pulse synchronization algorithm (whose more complex logic is likely to entail large delays). Second, by triggering the next "phase" when the previous one is completed, we avoid that the clocks are halted for a large period of time because the nodes wait for the next pulse before proceeding. This is for instance important if the system is supposed to respond quickly to external measurements or commands, disallowing long periods of "sleep".

Since we will show that the pulse algorithm stabilizes independently of the behavior of the NEXT signal, and the clock synchronization routine presented in Section 6 is designed such that it will stabilize once the pulse generation algorithm did so, we can partition the analysis of the compound algorithm into two parts. When proving the correctness of the pulse generation algorithm in Section 4, we thus assume that for each node i, NEXT_i is arbitrary.

3.1. Overview of the Algorithm

Before we describe the various state machines of the algorithm in detail, let us start with a conceptual overview of the algorithm that gives intuition on its structure and the purpose of its components. We also cover how the extension generating clocks we describe in more detail in Section 6 will couple to the pulse synchronization algorithm.

Each node runs several state machines that are organized in a layered structure and communicate by reading their state signals. On each layer, the state machines of the (at least $n - f$) non-faulty nodes cooperate in order to establish certain synchronization properties. The higher is a state machine in the hierarchy, the stronger are these guarantees; the lower it is, the weaker are the synchronization properties its input ports' signals need to satisfy for stabilization. The lowest-layer state machine utilizes randomization to recover from any configuration. Each other layer utilizes auxiliary information from the layer below to stabilize. Finally, we will "attach" another state machine on the top level that will output the logical clocks L_i.

More specifically, we have the following state machines:

— For the specific application of synchronizing logical clocks that we are going to describe in Section 6, at the top level each node employs a copy of the *quick cycle* state machine (Figure 16 in Section 6). Once all layers stabilized, the quick cycle behaves like a much faster version of the next lower layer, the main state machine, which generates the pulses. It relies on the pulses in order to ensure eventual stabilization. Once the system is stabilized, it consistently and deterministically increases the logical clock at a high frequency while guaranteeing small clock imprecision.

— The *main state machine* (Figure 3) is the centerpiece of the pulse generation algorithm. Once stabilized, it generates slow, roughly synchronized pulses within certain frequency bounds, by repeatedly switching to the state *accept*. These pulses are used as a "heartbeat" for stabilizing the application layer; at each pulse, the quick cycle's clocks are reset to 0 and the quick cycle's state machines are forced into state *accept*$^+$. By itself, however, the main state machine is not capable of recovering from *every* possible initial configuration of the non-faulty nodes. In certain cases, it requires some coarse synchrony to be established first in order to stabilize, which is probabilistically provided by the underlying layer. We remark that, once stabilized, the main state machine operates fully independently of this layer (and thus deterministically).

— The auxiliary information potentially required for stabilization by the main state machine is provided by a simple intermediate layer we refer to as *extension* of the main state machine (Figure 5). Essentially, it is supposed to be consistently reset by the underlying layer and then communicate information vital for stabilization to the main state machine. This information depends both on the time of reset and the current states of the n main state machines, which it therefore monitors.

— Finally, the *resynchronization routine* (Figure 6) utilizes randomized timeouts to consistently generate events at all non-faulty nodes that could be understood as "randomized pulses". Such a pulse is correct for our purposes if all non-faulty nodes generate a respective event in coarse synchrony and no non-faulty node generates another such event within a time window of a certain length. The crux of the matter is that an occasional occurrence of such pulses suffices to achieve stabilization deterministically. Relying on randomness on this layer greatly simplifies the task of overcoming the interference by faulty nodes at low costs in both time and communication. We note that the main state machine masks this randomness once stabilization is achieved, facilitating deterministic behavior of the higher levels and, ultimately, the application layer.

We will now present the individual state machines of the pulse synchronization algorithm in more detail; the discussion of the quick cycle is deferred to Section 6.

3.2. Basic Cycle

The full pulse generation algorithm makes use of a rather involved interplay between conditions on timeouts, states, and thresholds to converge to a safe state despite a lim-

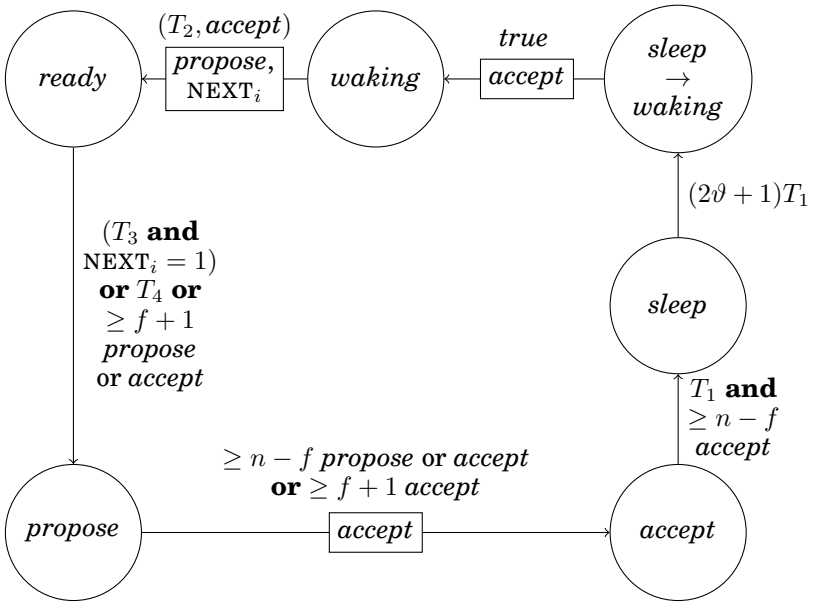

Fig. 1. Basic cycle of node i once the algorithm has stabilized.

ited number of faulty components. As our approach is thus complicated to present in bulk, we break it down into pieces. Moreover, to facilitate giving intuition about the key ideas of the algorithm, in this subsection we assume that there are never more than $f < n/3$ faulty nodes, i.e., the remaining $n - f$ nodes are non-faulty within $[0, \infty)$. We further assume that channels between non-faulty nodes (including loopback channels) are correct within $[0, \infty)$. We start by presenting the basic cycle that is repeated every pulse once a safe configuration is reached (see Figure 1).

We employ graphical representations of the state machine of each node $i \in V$. States are represented by circles containing their names, while transition $(s, s') \in \mathcal{T}$ is depicted as an arrow from s to s'. The guard $tr(s, s')$ is written as a label next to the arrow, and the reset function's value $re(s, s')$ is depicted in a rectangular box on the arrow. To keep labels more simple we make use of some abbreviations. Recall that in the notation of timeouts (T, s, C) the driving clock C is omitted. We write T instead of (T, s) if s is the same state which node i leaves if the condition involving (T, s) is satisfied. Threshold conditions like "$\geq f + 1$ s", where $s \in \mathbb{S}$, abbreviate Boolean predicates that reach over all of node i's memory flags $\mathrm{Mem}_{i,j,s}$, where $j \in V$, and are defined in a straightforward manner. If in such an expression we connect two states by "or", e.g., "$\geq n - f$ s or s'" for $s, s' \in \mathbb{S}$, the summation considers flags of both types s and s'. Thus, such an expression is equivalent to $\sum_{j \in V} \max\{\mathrm{Mem}_{i,j,s}, \mathrm{Mem}_{i,j,s'}\} \geq f + 1$. For any state $s \in \mathbb{S}$, the condition $S_{i,i} = s$, (respectively, $\neg(S_{i,i} = s)$) is written in short as "in s" (respectively, "not in s"). We write "true" instead of a condition that is always true (like e.g. "(in s) or (not in s)" for an arbitrary state $s \in \mathbb{S}$). Finally, $re(\cdot, \cdot)$ always requires to reset all memory flags of certain types, hence we write e.g. *propose* if all flags $\mathrm{Mem}_{i,j,\mathit{propose}}$ are to be reset.

We now introduce the basic flow of the algorithm once it stabilized (see Figure 2 for intuition; a more detailed explanation of the figure follows later), i.e., once all $n-f$ non-faulty nodes are well-synchronized, switching to state *accept* within $2d$. Recall that the remaining up to $f < n/3$ faulty nodes may produce arbitrary signals on their outgoing

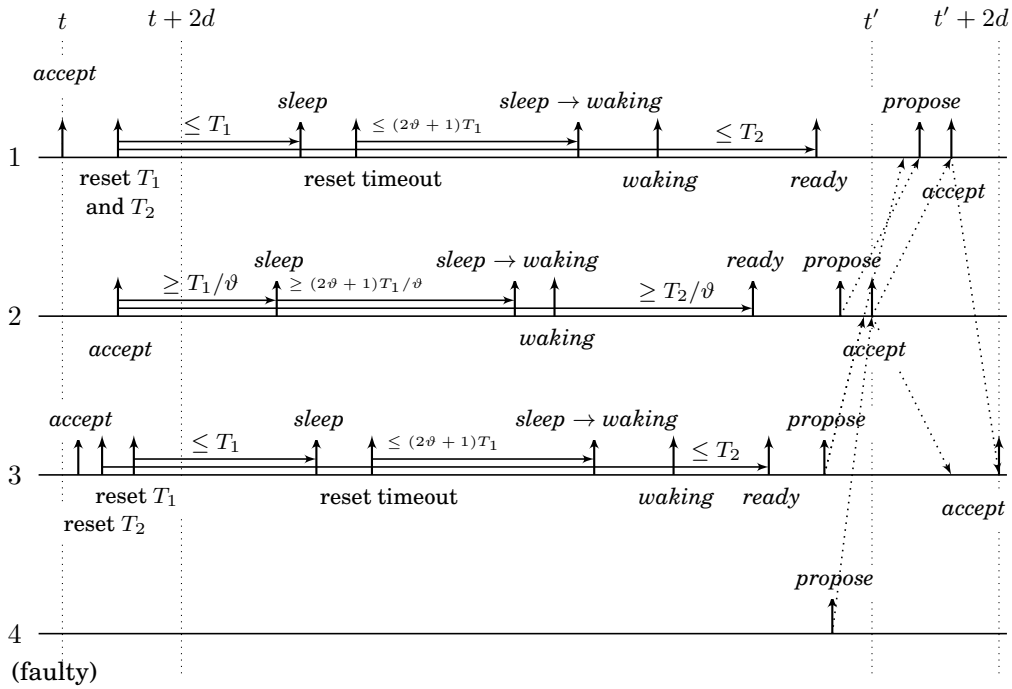

Fig. 2. Part of an execution of the basic cycle in a system of $n = 4$ nodes, where $f = 1$ node is faulty.

channels. A pulse is locally triggered by switching to state *accept*. Note that this starts two timeouts, $(T_1, \textit{accept})$ and $(T_2, \textit{accept})$. Assume that at some time all non-faulty nodes switch to state *accept* within a time window of $2d$, i.e., pulses are generated by non-faulty nodes within a time interval of size $2d$. Supposing that $T_1 \geq 3\vartheta d$, these nodes will observe, and thus memorize, each other and themselves in state *accept* within a time interval of size $3d$ and thus before T_1 expires at any non-faulty node for the following reasons: Assuming that the first non-faulty node switches to *accept* at time t, its timeout $(T_1, \textit{accept})$ is reset immediately when switching to *accept* at earliest and thus does not expire before $t + T_1/\vartheta$. It thus cannot switch to another state before time $t + T_1/\vartheta$. The last non-faulty node, by assumption, will switch to *accept* by time $t + 2d$. Its state will be propagated to all non-faulty nodes by at most another d later. By the constraint $T_1 \geq 3\vartheta d$, at that time, all non-faulty nodes are still in state *accept*.

Hence, the nodes will switch to state *sleep* upon expiration of their T_1 timeouts. From state *sleep*, they will switch to states *sleep* → *waking*, *waking*, and finally *ready*, where the timeout $(T_2, \textit{accept})$ is determining the time this takes, as it is considerably larger than $\vartheta(2\vartheta + 2)T_1$. The intermediate states serve the purpose of achieving stabilization, hence we leave them aside for the moment.

Nodes in *ready* will switch to *propose* as soon as (i) they memorize $f + 1$ nodes in *propose* or *accept*, (ii) T_3 is expired and NEXT_i is memorized as having been observed in state 1, or (iii) T_4 expires. Using memory flags instead of the immediate signals here is vital in order to avoid that faulty nodes can induce metastability, e.g. by switching back a *propose* signal to 0 just when the $(f + 1)$-threshold is reached, or continuously

maintaining a voltage on the channel that cannot be clearly interpreted as either 0 or 1.[20]

Assume for a moment that all non-faulty nodes are observed in state *ready* with all their *propose* and *accept* flags reset to 0 before the first node switches to *propose*; we will show later that this is indeed the case. Thus, the first non-faulty node that switches to state *accept* again cannot do so before it memorizes at least $n - f$ nodes in state *propose*. This implies that at the time when it switches to *accept*, at least $n - 2f \geq f + 1$ non-faulty nodes must be in state *propose*. Hence, the rule that nodes switch to *propose* if they memorize $f + 1$ nodes in states *propose* or *accept* will take effect, i.e., the remaining non-faulty nodes in state *ready* switch to *propose* after less than d time. Another d time later all non-faulty nodes in state *propose* will have become aware of this and switch to state *accept* as well, as the threshold of $n-f$ nodes in states *propose* or *accept* is reached. Thus the cycle will be complete and the reasoning can be repeated inductively.

It remains to show that indeed the memory flags are properly reset and nodes are observed in state *ready* before a non-faulty node switches to *propose*. To see this, recall that by time $t + T_1 + 4d$, no non-faulty node is observed in state *accept* (or *propose*, for that matter) anymore until the first node switches to *propose* again. Checking the timeout conditions for the transition from *sleep* to *sleep* $\rightarrow$ *waking* and recalling that $T_1 \geq 3\vartheta d$, we see that no non-faulty node switches to *waking* earlier than time $t + (2 + 1/\vartheta)T_1 > t + T_1 + 4d$. Hence, no node will memorize a non-faulty node in *propose* or *accept* after resetting these flags upon switching to *waking* and *ready* (and before a node switches to *propose* again). This entails that the first node switching to *propose* must do so because T_4 expired or because T_3 expired and its NEXT memory flag is true. As all nodes switched to *accept* during $[t, t+2d)$, at non-faulty nodes the timeouts $(T_2, \textit{accept})$ and subsequently $(T_3, \textit{ready})$ or $(T_4, \textit{ready})$ cannot expire again before time $t + (T_2 + \min\{T_3, T_4\})/\vartheta$. On the other hand, each non-faulty node switches to *accept* by time $t + 2d$ and hence resets T_2 by time $t + 3d$. It will expire by time $t + T_2 + 3d > t + (2\vartheta + 2)T_1 + 6d$ (the latest possible time when a node observes itself in *waking*), making it switch to *ready*, in which it will be observed by time $t + T_2 + 4d$. Therefore, the constraint $(T_2 + \min\{T_3, T_4\})/\vartheta \geq T_2 + 4d$ is sufficient to ensure that indeed all non-faulty nodes observe themselves in state *ready* before the first one switches to *propose*.

Clearly, for the above line of argumentation to be valid, the algorithm could be simpler than stated in Figure 1. We already mentioned that the motivation of having three intermediate states between *accept* and *ready* is to facilitate stabilization. Similarly, there is no need to make use of the *accept* flags in the basic cycle at all; in fact, it adversely affects the constraints the timeouts need to satisfy for the above reasoning to be valid. However, the *accept* flags are much better suited for diagnostic purposes than the *propose* flags, since nodes are expected to switch to *accept* in a small time window and remain in state *accept* for a small period of time only (for all our results, it is sufficient if $T_1 = 4\vartheta d$). Moreover, two different timeout conditions for switching from *ready* to *propose* are unnecessary for correct operation of the pulse synchronization routine. The second timeout is introduced in combination with the NEXT signal in order to allow for a seamless coupling to the application layer.

[20]These metastability-related issues are not covered by Definition 2.1, as they are to be addressed by an implementation of the state machine representation utilized in this work; a discussion is beyond the scope of this article. In contrast, Definition 2.1 focuses on state transitions where branching is possible, e.g., from *accept* to either *recover* or *sleep* in Figure 3, which require to enable a metastability-free implementation by careful algorithmic design beyond mere use of memory flags.

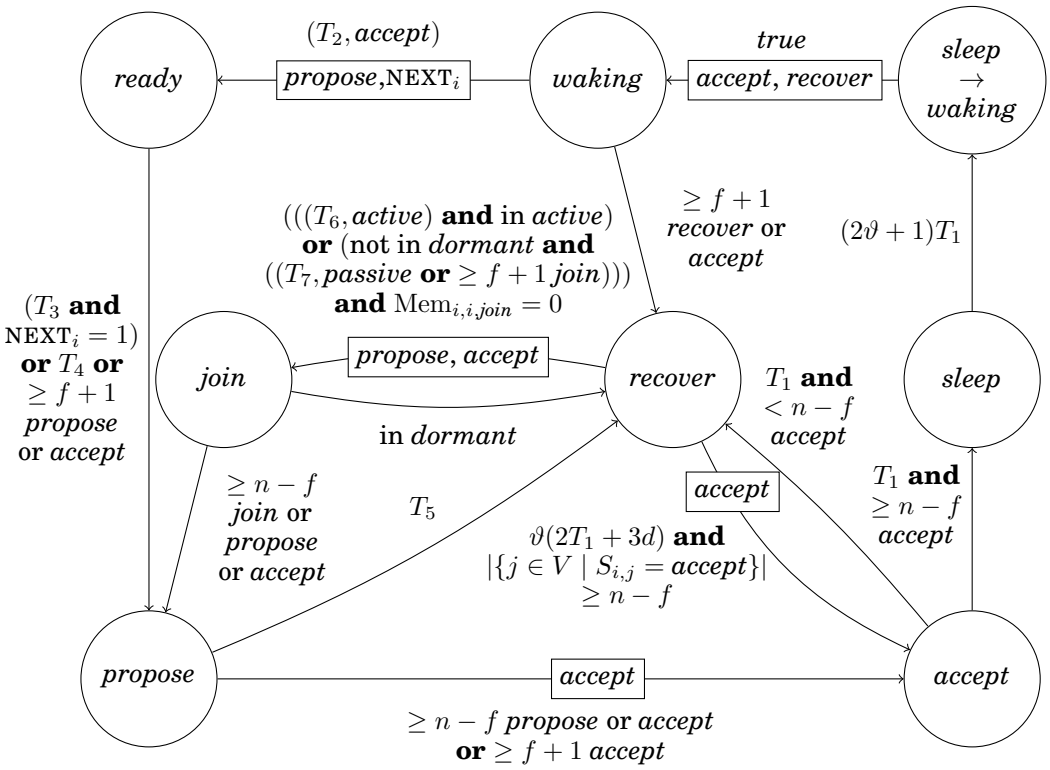

Fig. 3. Overview of the core routine of node i's self-stabilizing pulse algorithm.

Figure 2 shows part of an execution of the basic cycle executed by four nodes, one of them being faulty. Communication delays between nodes (at the time when the sending node switches state) are depicted by dotted arrows. Note however, that a node continuously transmits its state, and not only when switching state. In order not to overload the figure we only drew those arrows responsible for the receiving node making a state transition and did not draw transmissions of a node to itself. An exception is node 4 (the faulty node) that only sends its state correctly to node 2, making it switch to *accept* earlier than the other nodes. Node 1 is slow, in the sense that it has slow clocks and (the maximum of) d time elapses between the node switching to a new state and node 1 resetting its timeouts (e.g., between switching to *accept* and resetting timeouts $(T_1, accept)$ and $(T_2, accept)$). Node 2 is fast: it immediately resets its timeouts and runs a fast clock. Nodes 2 and 3 switch to *propose* because of their timeouts $(T_3, ready)$ or $(T_4, ready)$ expiring. The slow node 1, however, switches to *propose* because it memorized $f+1 = 2$ nodes in *propose*, namely, nodes 2 and 3. Observe that all non-faulty nodes switch to *accept* within $2d$ time again: Node 1 switches to *accept* because it memorized itself, node 2 and node 3 in *propose*, and node 2 in *accept*, which is more than the required $2f+1 = 3$ nodes in *propose* or *accept*. Node 2 switches to *accept* because it memorizes itself and nodes 3 and 4 in *propose*, and node 3 because it memorizes nodes 1 and 2 in *accept*.

3.3. Main Algorithm

We proceed by describing the main routine of the pulse algorithm in full. It is graphically presented in Figure 3. Except for the states *recover* and *join* and additional resets

of memory flags, the main routine is identical to the basic cycle. The purpose of the two additional states is the following: Nodes switch to state *recover* once they detect that something is wrong, that is, non-faulty nodes do not execute the basic cycle as outlined in Section 3.2. This way, non-faulty nodes will not continue to confuse others by sending for example state signals *propose* or *accept* despite clearly being out-of-sync. There are various consistency checks that nodes perform during each execution of the basic cycle. The first one is that in order to switch from state *accept* to state *sleep*, non-faulty nodes need to memorize at least $n - f$ nodes in state *accept*. If this does not happen within $4d \leq T_1/\vartheta$ time after switching to state *accept*, by the arguments given in Section 3.2, the nodes could not have entered state *accept* within $2d$ of each other. Therefore, any node making this observation can be certain that something is wrong and switches to state *recover* upon expiration of T_1. Next, whenever a non-faulty node is in state *waking*, there should be no non-faulty nodes in states *accept* or *recover*. Considering that the node resets its *accept* and *recover* flags upon switching to *waking*, it follows that no node should memorize $f + 1$ or more nodes in states *accept* or *recover* at any time when it observes itself in state *waking* (even if it never *observes* this many nodes in either of the states). If it does, however, it again switches to state *recover*. Last but not least, during a synchronized execution of the basic cycle, no non-faulty node may be in state *propose* for more than a certain amount of time before switching to state *accept*. Therefore, nodes will switch from *propose* to *recover* when timeout T_5 expires.

There are three basic stabilization scenarios by which correct operation may be restored:

(i) All non-faulty nodes are in state recover or switch to this state due to the consistency checks described above. Eventually, they will switch to state *join* due to a timeout mechanism, and essentially the same voting scheme employed in the basic cycle ($\geq f + 1$ nodes memorized in *join* "pull" others into *join*, and a threshold of $n - f$ nodes memorized in *join* or its immediate successor states are required to switch to *propose*) is utilized to generate a clean pulse that "reboots" the system.

(ii) At least $n - 2f \geq f + 1$ nodes keep executing the basic cycle in a synchronized fashion. The consistency checks of the basic cycle together with the requirement that $n - f$ nodes must be observed in *accept* within a short time interval (roughly T_1) in order to switch from *accept* to *sleep* make sure that a smaller number will inevitably lead to scenario (i). Having $f + 1$ or more non-faulty nodes switch to *sleep* $\rightarrow$ *waking* in a short time period then is used to establish (rough) synchrony among *all* non-faulty nodes. Starting certain timeouts at such a *resynchronization point* (see Definitions 3.1 and 3.2) and choosing the timing constraints accordingly, we ensure that the following sequence of transitions occurs:

— All non-faulty nodes on the basic cycle switch to either *ready* or *recover*.
— All non-faulty nodes in *recover* switch to *join* before any node in state *ready* could possibly switch to *propose* due to an expired timeout.
— In all possible cases, the two "pulling" rules involving $(f + 1)$-thresholds guarantee that all non-faulty nodes will switch to *accept* within $3d$ time, at the latest once the T_4 timeouts expire and all non-faulty nodes switched to states *join* or *propose*.

(iii) At least $n - f$ nodes execute the basic cycle in a synchronized fashion. Any further node will either start following the basic cycle as well or end up in *recover*, from where it will "jump on the train" by switching to *accept* when the majority of nodes switches to *accept* on the next pulse. Scenarios (i) and (ii) already suffice to recover from arbitrary initial states, but require time $\Omega(n)$ in the worst case, as we will discuss later. Scenario (iii) can be integrated without affecting Scenarios (i) or (ii) and succeeds within $\mathcal{O}(1)$ time.

We point out that Scenario (i) (or something similar) cannot be avoided if we want to have a recovery mechanism where nodes leave the basic cycle,[21] as otherwise the system could deadlock. On the other hand, Scenario (ii) is necessary because nodes must never wait for observing more than $n - f$ state transitions when executing the basic cycle, as otherwise the system would deadlock if f or more nodes crash. Hence, f Byzantine nodes can always mimic to be non-faulty to $n-2f$ non-faulty nodes executing the basic cycle in order to prevent them from switching to *recover*; these $n - 2f$ nodes cannot distinguish this setting from f faulty nodes claiming to be in state *recover* and $n - f$ nodes properly advancing on the basic cycle and therefore must not leave it.

These considerations reveal that we need to reconcile two conflicting goals: We must be able to "reboot" the system after a "complete crash", which, given that initial states are arbitrary, cannot rely on any assumptions on synchrony, yet we must also be able to make a minority of nodes in *recover* accept the timing imposed by $f + 1$ or more non-faulty nodes that execute the basic cycle and cannot detect that the system is not operating correctly. The difficulty here is that it is vital that all non-faulty nodes decide consistently on a common time base or Scenario (i) applies.

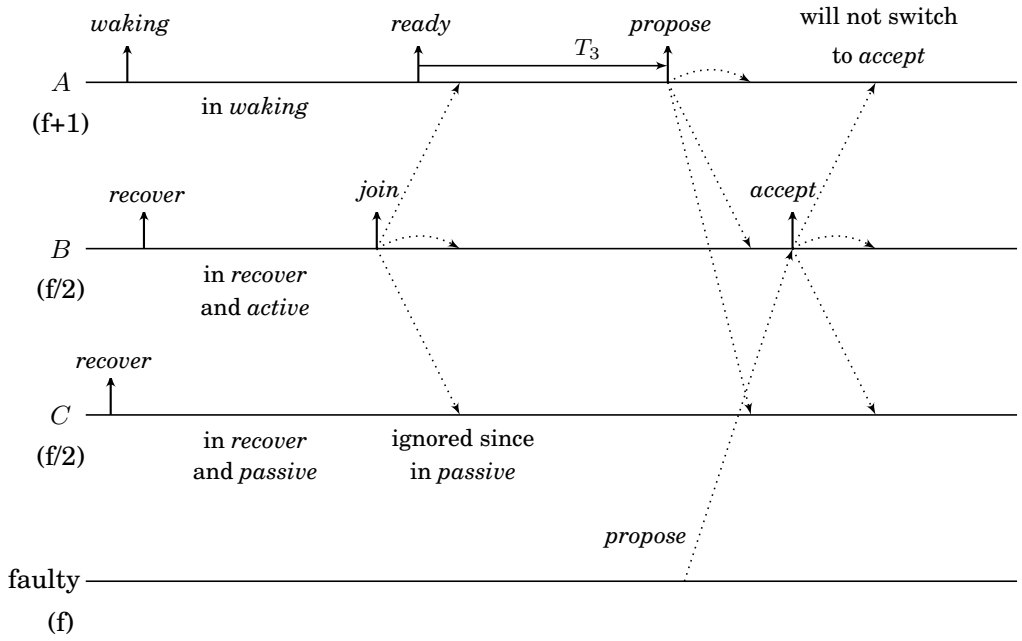

Fig. 4. Part of an execution where nodes are prevented to join the basic cycle in synchrony.

To get some intuition on this issue, consider the partial execution of the main state machine given in Figure 4. The states *active* and *passive* of the extension of the main state machine, which will be introduced shortly, keep track of a node's observation on a resynchronization point. For now, it is enough to know that a node will switch from *passive* to *active* (and therefore start timeout T_6) when it notices $n - f$ nodes switching to *sleep* $\rightarrow$ *waking*, whereas T_7 is a large timeout that leaves sufficient time for the "synchronized" recovery mechanism to work before we fall back to Scenario (i). The

[21] Not employing such an approach turns out to be even more difficult, as the nodes' behavior on the basic cycle is constrained by the properties we expect from the stable system.

execution in Figure 4 describes a setting where at least $f+1$ non-faulty nodes execute the basic cycle in synchrony (set A, initially in state *waking*), at least f nodes initially are in *recover* (set $B \cup C$), nodes in set B are in state *active*, and nodes in set C are in state *passive*; in particular, the condition "not in *dormant*" is satisfied. All memory flags are cleared and all timeouts have just been reset.

The nodes in A will switch to *ready* (the faulty nodes will not send *recover* or *active* signals to them at this stage), and we assume that set B will switch to *join* because T_6 expires as prescribed by Scenario (ii). Nodes in C are not in *active* and therefore will not switch to *join* (as the faulty nodes do not send *join* signals) anytime soon. When the nodes in A switch to *propose*, the nodes in C will thus not participate in the voting process, implying that the faulty nodes have full control on which nodes from $A \cup B$ switch to *accept* and when (in Figure 4, faulty nodes only send to B making them switch to *accept*, whereas nodes in C remain in *recover*). For instance, by careful control of non-faulty nodes' clocks (within the valid bounds) and faulty nodes communication, the adversary can arrange for some non-faulty nodes switching from *propose* to *recover* because they have been stuck in *propose* until T_5 expires, exactly at the time when at the nodes in C timeout T_7 expires and they switch to *join* and subsequently to *propose*.

While this example does not provide a complete non-stabilizing execution, it shows a behavior of the system that certainly is to be shunned. If the adversary gains too much control of when a subset of the nodes rejoins the execution of the basic cycle, this can be used to make some nodes leave the basic cycle just when others enter it again. This way, the adversary can maintain a split of the system, where always subsets of the nodes follow different stabilization strategies that require different timing, and ultimately leverage this to prevent stabilization. Any individual such scenario is not very illustrative, as typically it can be resolved by a simple modification of the main state machine. However, when devising the algorithm, we encountered the problem that any such attempt just made the same issue reappear for another execution and another combination of states.

In the example from Figure 4, the underlying issue was that sets B and C had differing views whether a common timebase had been established by the nodes still executing the basic cycle. Once such a common timebase can indeed be established, it can be used to correctly "initialize" any states and memory flags that do not interact with the basic cycle in case all non-faulty nodes already execute the basic cycle in a synchronized fashion. The design of the main state machine thus can be seen as a way to separate the problem of stabilization sufficiently from the indefinite execution of the basic cycle such that it can be solved by an "attached" algorithm that terminates within bounded time, and provide a way to properly initialize and run this algorithm every now and then. This is the purpose of the remaining state machines we have not discussed yet.

3.4. Extension of the Main State Machine

The *extension* of the main state machine (Figure 5) controls the execution of the implicit stabilization algorithm. Its three states correspond to the stabilization algorithm not running (*dormant*), running without having established a common time base so far (*passive*), and running with a common time base (supposedly) being established by having observed at least $f+1$ nodes in state *sleep* $\rightarrow$ *waking* since the algorithm was started. The latter is captured by the *sleep* $\rightarrow$ *waking* flags that are reset upon starting a stabilization attempt, which in turn is controlled by the state *resync* of the resynchronization algorithm that will be introduced shortly. The *join* flags are also reset when switching to *passive*, as they will be required only once during a stabilization attempt. Note that the nodes cannot enter state *join* of the main state machine and are forced to leave it whenever in *dormant*, ensuring that the *join* flags will be properly cleared

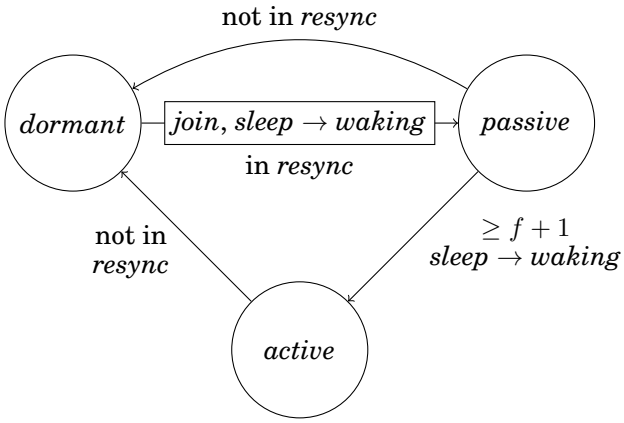

Fig. 5. Extension of node i's core routine.

when initializing stabilization. We formalize the notion of "initializing stabilization" by the following definition, which incorporates the predecessor state of *resync* in the resynchronization state machine.

Definition 3.1 (*Resynchronization Points*). Given $W \subseteq V$, time t is a W*-resynchronization point* iff each node in W switches to state *supp* $\rightarrow$ *resync* in the time interval $(t, t + 2d)$.

Despite the simplicity of the control structure the extension state machine provides, it is in general not trivial to guarantee a correct initialization even if all non-faulty nodes switch to *resync* at the same instant in time and remain in this state sufficiently long for T_7 to expire (which essentially is equivalent to termination of the implicit stabilization algorithm). In the unfortunate event that a resynchronization point is contained in or close to the small time window during which different non-faulty nodes executing the basic cycle may be observed in state *sleep* $\rightarrow$ *waking*, some nodes may immediately switch to *active* while others remain in *passive*. A sufficient condition to avoid this is to assume that no node is in state *sleep* during a specific time window around a resynchronization point t. A similar issue arises if nodes are still in state *join* very close to a resynchronization point (i.e., they have not already been in *dormant* for a while), as this may interfere with a clean reset of the *join* flags. Our proof of stabilization will rely on resynchronization points without these caveats, which is expressed by the following definition.

Definition 3.2 (*Good Resynchronization Points*). A W-resynchronization point at time t is called *good* iff no node from W switches to state *sleep* during $(t - \Delta_g, t)$, where $\Delta_g := (2\vartheta + 3)T_1 + 2d$, and no node is in state *join* during $[t - T_1 - 2d, t + 4d)$.

It remains to explain how stabilization is properly initialized, or, equivalently, how we ensure that good resynchronization points occur within $\mathcal{O}(n)$ time with a large probability, regardless of the initial state of the system.

3.5. Resynchronization Algorithm

The resynchronization routine is specified in Figure 6. It provides some synchronization that is akin to that of a pulse, except that such "weak pulses" occur at random times, and may be generated inconsistently even after the algorithm as a whole has stabilized. Since the main routine operates independently of the resynchronization routine once the system has stabilized, we can afford the weaker guarantees of the

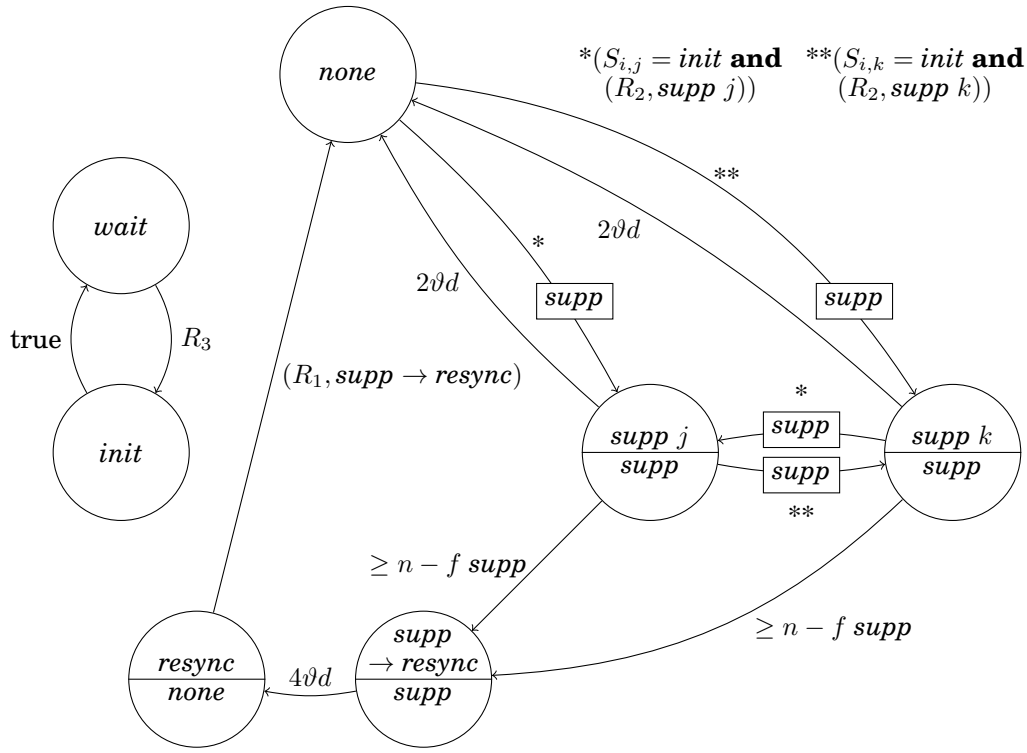

Fig. 6. Resynchronization algorithm, comprising two state machines executed in parallel at node i. The states *supp* j and *supp* k (as well as the respective transitions) are generic examples of the states *supp* 1 to *supp* n comprised by the state machine

routine: If it succeeds in generating a good resynchronization point merely once, the main routine will stabilize deterministically.

In order to clarify that despite having a linear number of states ($supp_1, \ldots, supp_n$), this part of the algorithm can be implemented using 2-bit communication channels between state machines only, we generalize our description of state machines as follows. If a state is depicted as a circle separated into an upper and a lower part, the upper part denotes the local state, while the lower part indicates the signal state to which it is mapped. A node's memory flags then store the respective signal states only, i.e., remote nodes do not distinguish between states that share the same signal. Clearly, such a machine can be simulated by a machine as introduced in the model section. The advantage is that such a mapping can be used to reduce the number of transmitted state bits; for the resynchronization routine given in Figure 6, we merely need two bits (*init*/*wait* and *none*/*supp*) instead of $\lceil \log(n+3) \rceil + 1$ bits.

The basic idea behind the resynchronization algorithm is the following: Every now and then, nodes will try to initiate agreement on a resynchronization point. This is the purpose of the small state machine on the left in Figure 6, which makes the node briefly switch to *init* once its randomized timeout R_3 expires and then switch back again. Assume for the moment that each node will do so constantly often in $\Theta(n^2)$ time, and between each two such attempts it will let $\Omega(n^2)$ time pass. Hence, it is safe to ignore any initialization message by a node that sent such a message recently, and each node will "listen" to a faulty node at most $\mathcal{O}(1)$ times within an interval of length $\Theta(n^2)$. Therefore, faulty nodes can interfere at most $\mathcal{O}(n^2)$ many times with stabiliza-

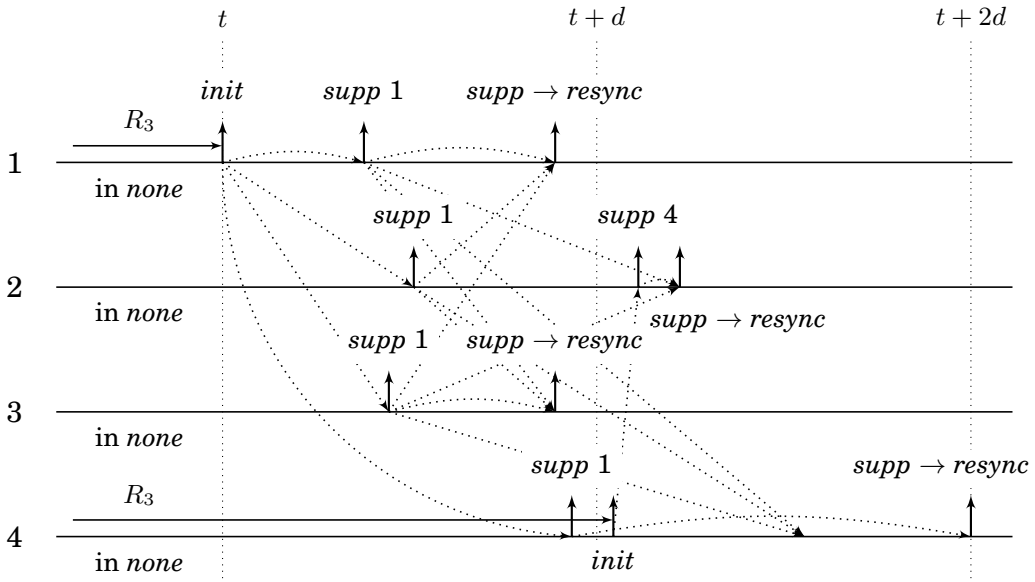

Fig. 7. Part of an execution of the resynchronization algorithm in a system of $n = 4$ nodes in the absence of faulty nodes.

tion attempts of non-faulty nodes within $\mathcal{O}(n^2)$ time. Choosing constants accordingly and taking into account that the adversary cannot predict when a randomized timeout will expire, we can make sure that there is a large probability that a non-faulty node will initialize stabilization at some time when faulty nodes do not interfere; here, it is vital to ensure that once the expiration of a timeout is revealed (i.e., an *init* signal is triggered), it is already too late for the faulty nodes to prevent a good resynchronization point. We remark that the use of randomness serves not only to overcome the faulty nodes' interference, but conveniently makes it likely that a resynchronization point is good, because most of the time nodes will be (a) in state *none* (and therefore in *dormant* and not in *join*) and (b) not in state *sleep* of the basic cycle.

The resynchronization algorithm in Figure 6 follows the same principle, but refines it by an additional voting step in order to reduce the time bound from above (and thus also the overall stabilization time) from $\Theta(n^2)$ to $\Theta(n)$. By requiring $n - f$ *supp* signals in order to switch to *supp* $\rightarrow$ *resync*, we force the adversary to "invest" at least $n - 2f > n/3$, i.e., a linear number of *init* signals from faulty nodes to non-faulty nodes in order to convince a non-faulty node to switch to *supp* $\rightarrow$ *resync* (and subsequently *resync*). Any smaller number of signals will not interfere with a non-faulty node initializing resynchronization, as nodes in *supp* states still react to *init* signals by other nodes.

Consider now the state machine displayed on the right of Figure 6. We will illustrate how the routine is intended to work. Figure 7 shows part of an execution discussed in the following, in the case of a four node system. Assume that at the time t when a non-faulty node i (node 1 in the figure) switches to state *init*, all non-faulty nodes are not in any of the states *supp* $\rightarrow$ *resync*, *resync*, or *supp* i, and at all non-faulty nodes the timeout $(R_2, \textit{supp } i)$ has expired. Then, no matter what the signals from faulty nodes or on faulty channels are (in the figure all nodes are non-faulty), each non-faulty node will be in one of the states *supp* j, $j \in V$, or *supp* $\rightarrow$ *resync* at time $t + d$. Hence, they will observe each other (and themselves) in one of these states at some time smaller

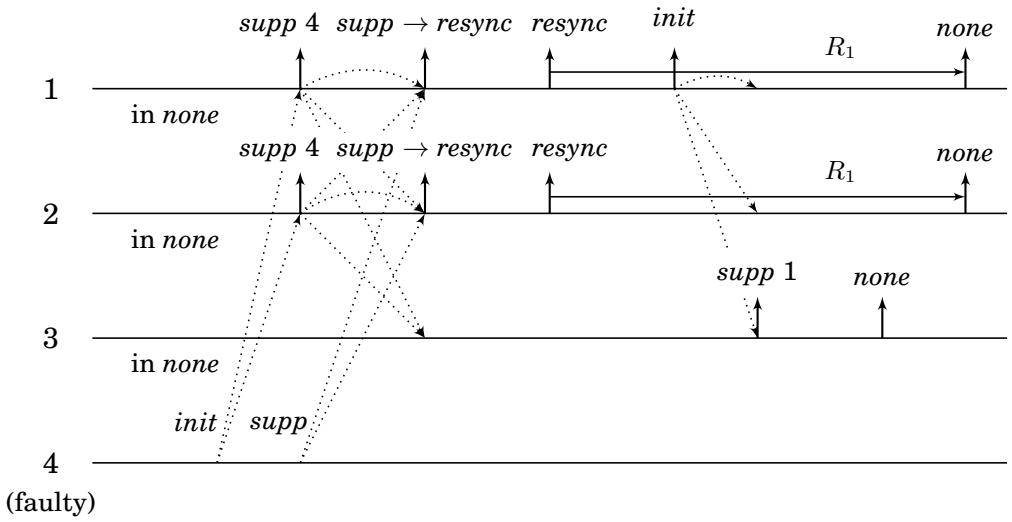

Fig. 8. The adversary spoiling an attempt to generate a resynchronization point in a system of $n = 4$ nodes.

than $t + 2d$. These statements follow from the various timeout conditions of at least $2\vartheta d$ and the fact that observing[22] node i in state *init* will make nodes switch to state *supp* i if in *none* or *supp* j, $j \neq i$. Hence, all of them will switch to state *supp* $\rightarrow$ *resync* during $(t, t + 2d)$, i.e., t is a resynchronization point. Note that timeout R_1 makes sure that no non-faulty node will leave *resync* again too soon, leaving sufficient time for the main routine to stabilize.

The scenario we just described relies on the fact that at time t no non-faulty node is in state *supp* $\rightarrow$ *resync* or state *resync*, and $(R_2, \textit{supp}\ i)$ is expired at all non-faulty nodes. Figure 8 shows how the adversary can use inconsistent communication to impede the stabilization process by preventing that a non-faulty node's transition to *init* will trigger a resynchronization point. We will choose $R_3/\vartheta > R_2 \gg R_1$, implying that when R_3 expires at node i again after it switched to state *init*, $(R_2, \textit{supp}\ i)$ is indeed expired at all nodes and they switched back to state *none* (unless other *init* signals interfered). With this in mind, we can see that the requirement that non-faulty nodes are neither in *supp* $\rightarrow$ *resync* nor *resync* is achieved with a large probability by choosing R_3 as a uniform distribution over some interval $[\vartheta(R_2 + 3d), \vartheta R_2 + \Theta(nR_1)]$: Other nodes will switch to *init* $\mathcal{O}(n)$ times during this interval, each time "blocking" other nodes for at most $\mathcal{O}(R_1)$ time, and, by the arguments given above, faulty nodes may interfere at most $\mathcal{O}(n)$ times in total as well, also "blocking" at most $\mathcal{O}(R_1)$ time. If the random choice picks any other point in time during this interval, a resynchronization point occurs. Even if the clock speed of the clock driving R_3 is manipulated in a worst-case manner (affecting the density of the probability distribution with respect to real time by a factor of at most ϑ), just increasing the size of the interval further accounts for this.

We emphasize that the above strategy fails if the adversary can predict when a timeout is going to expire, which is sketched in Figure 9 for $n = 7$ and $f = 2$. Initially, all 5 non-faulty nodes are in state *none* with all timeouts R_1 and R_2 expired. The timeouts R_3 have just been reset, and the "random" choices determined that they expire

[22] Since nodes will act immediately when seeing an *init* signal, utilizing memory flags here would be pointless.

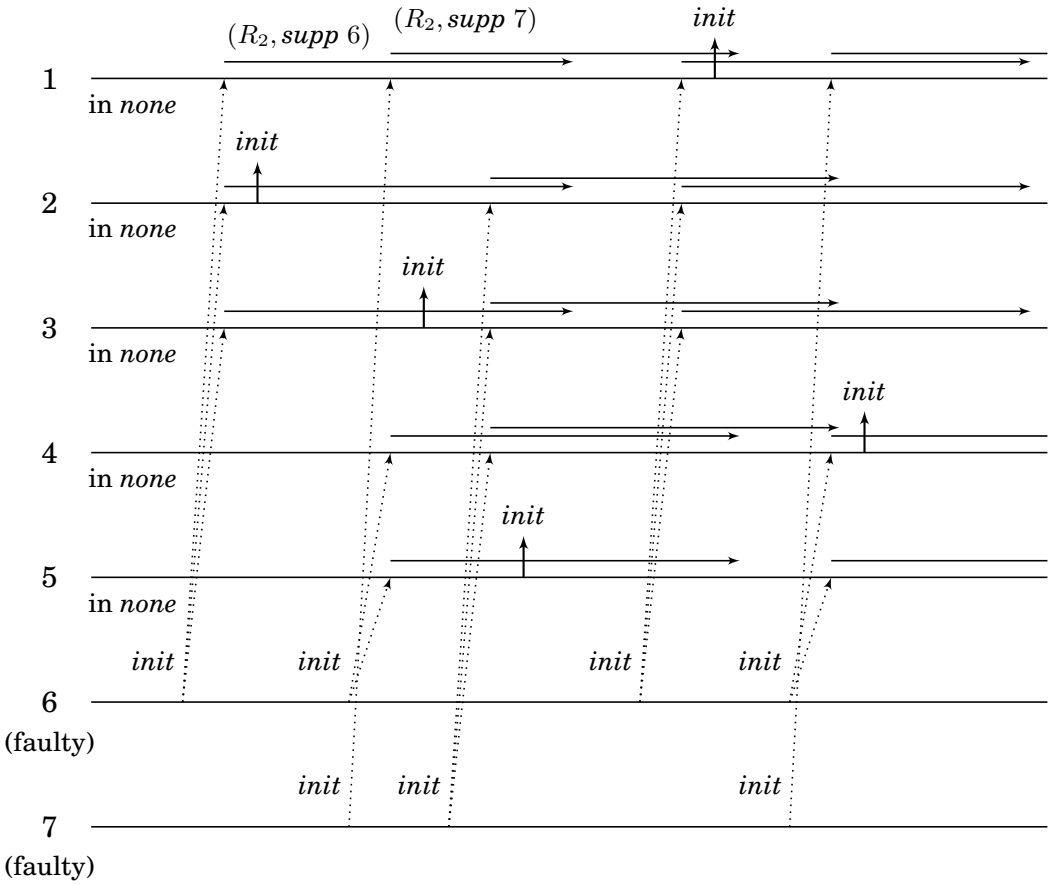

Fig. 9. The adversary preventing stabilization for $\Theta(n)$ time in a system of $n = 7$ nodes. We only show the *init* signals by faulty nodes, transitions to *init*, and R_2 timeouts relating to faulty nodes here; for each switch of a non-faulty node to *init*, the adversary follows the approach shown in Figure 8 to foil occurrence of a resynchronization point. The pattern can be repeated unless the non-faulty nodes "get lucky" in that their R_3 timeouts line up neatly (i.e., 4 nodes switch to *init* in a time window to narrow for R_2 to expire, yet there is sufficient time in between for R_1 to expire and previously fooled nodes to switch back to *none*).

at almost equidistant times, i.e., the random choices produced a well-spread and thus good distribution from the point of view of the algorithm. However, since the adversary knows the respective times in advance, he can just repeat the strategy shown in Figure 8 for each time when a non-faulty node switches to *init*, never "wasting" any *init* signals to non-faulty nodes.

Concretely, shortly before a non-faulty node i sends an *init* signal at time t, say, at time $t-10\vartheta d$, the faulty nodes will send an *init* signal on $f+1 = 3$ channels (to different non-faulty nodes), causing the recipients to switch to some *supp* state. Furthermore, both of them sent a *supp* signal to these nodes, causing them to switch to *supp* $\rightarrow$ *resync* and subsequently *resync* before the *init* signal is broadcasted by the respective non-faulty node. The remaining two non-faulty nodes that are in state *none* at time t will reset their *supp* flags upon switching to *supp* i when receiving the *init* signal from i. Hence, they will not proceed to *supp* $\rightarrow$ *resync* because the threshold "$\geq n-f = 5$ *supp*" cannot be reached within $2\vartheta d$ local time, and they will switch back to *none*.

Since there are 5 attempts to spoil, each of which requires 3 *init* signals from faulty to non-faulty nodes, the adversary can repeat the above strategy by sending at most 2 *init* signals over each of the $2 \cdot 5$ channels between faulty and non-faulty nodes within the considered period of time. Simple computations reveal that the adversary can in fact prevent a good resynchronization point indefinitely, provided that the local time when a randomized timeout will expire becomes known when the timeout is started; the scenario can be generalized to arbitrarily large n, $n = 3f + 1$.[23]

Given that we assume a strong adversary that knows the entire state of the system at all times, it is thus vital to meet the independence condition on the random choices stated in the definition of randomized timeouts. In particular, one must not "pre-evaluate" the distribution when the timeout is started. Rather, whenever a local time is reached where the timeout *may* expire, a random choice is made deciding only whether the timeout indeed does expire or will do so later. On the other hand, in practice it can very well be feasible to simply draw from the random distribution when the timeout is started, since in many settings it is reasonable to assume that the expiration time of the timeout is revealed to the remaining system no sooner than it actually affects the respective node's behavior. Moreover, employing pseudo-randomness will be sufficient unless the system is to be resilient against an actual attacker; it seems absurd to assume that a fault pattern that predicts future pseudo-random choices will emerge naturally.

3.6. Timeout Constraints

Condition 3.3 summarizes the constraints we require on the timeouts for the core routine and the resynchronization algorithm to act and interact as intended.

Condition 3.3 (*Timeout Constraints*). Recall that $\vartheta > 1$ and $\Delta_g := (2\vartheta + 3)T_1 + 2d$. Define

$$\lambda := \sqrt{\frac{25\vartheta - 9}{25\vartheta}} \in \left(\frac{4}{5}, 1\right) . \tag{1}$$

The timeouts need to satisfy the constraints

$$\begin{aligned}
T_1 &\geq 4\vartheta d && (2)\\
T_2 &\geq 3\vartheta\Delta_g + 7\vartheta d = (6\vartheta^2 + 9\vartheta)T_1 + 13\vartheta d && (3)\\
T_3 &\geq (2\vartheta^2 + 4\vartheta)T_1 - T_2 + \vartheta T_6 + 7\vartheta d \overset{(2,7)}{>} (\vartheta - 1)T_2 + 6\vartheta d && (4)\\
T_4 &\geq T_3 && (5)\\
T_5 &\geq \max\{(\vartheta - 1)T_2 - T_3 + \vartheta T_4 + 7\vartheta d, (\vartheta - 1)T_1 + \vartheta(T_2 + T_4) - T_6\} && (6)\\
T_6 &\geq \vartheta T_2 - 2\vartheta T_1 + 2\vartheta d \overset{(3)}{>} (2\vartheta^2 + 3\vartheta)T_1 + 6\vartheta d && (7)\\
T_7 &\geq (4\vartheta - 2)T_1 + \vartheta(T_2 + T_4 + T_5) + T_6 + 2\vartheta d\\
&\overset{(7)}{>} (2\vartheta^2 + 7\vartheta - 2)T_1 + \vartheta(T_2 + T_4 + T_5 + 8d) && (8)
\end{aligned}$$

[23] We remark that in case ϑ is not too large, over every interval of $\Theta(n)$ time, there is a positive probability that the times when the non-faulty nodes' timeouts R_3 expire are distributed such that the adversary, knowing them in advance, still cannot prevent stabilization. However, Chernoff's bound shows that the probability for this to happen is exponentially small in $n-f$; hence, eventual stabilization may be guaranteed with probability 1 even for an omniscient adversary, but the expected stabilization time is exponential in n.

$$R_1 \geq \max\left\{\vartheta T_7 + (4\vartheta^2 + 8\vartheta)d, \vartheta(2T_1 + 2T_2 + 2T_4 + T_5 + 12d) - 2T_1\right\} \tag{9}$$

$$R_2 \geq \frac{2\vartheta(R_1 + 6\Delta_g + T_1 + (8\vartheta + 11)d)(n - f)}{1 - \lambda} \tag{10}$$

$$R_3 = \text{uniformly distributed random variable on } [\vartheta(R_2 + 3d), \vartheta(R_2 + 3d) + 8(1 - \lambda)R_2] \tag{11}$$

$$\lambda \leq \frac{T_2 - 2\vartheta\Delta_g - (\vartheta - 1)T_1 - 2\vartheta d}{T_2 - (\vartheta - 1)T_1 - \vartheta d}\ . \tag{12}$$

We need to show that this system can always be solved. Furthermore, we would like to be able to couple the pulse generation algorithm to application algorithms that need to control the timing of their own progression (once stabilization is achieved), regardless of how much the time span for the tasks such an algorithm performs between successive pulses may vary (within known bounds). More concretely, we would like to couple our algorithm with a simple clock synchronization algorithm presented in Section 6. Its idea is to let each node generate $M \geq 1$ fast pulses between any two FATAL pulses. For this algorithm to work correctly, we must account for the possible drift of the fast M-clock and the desynchronization of the FATAL pulses, making sure that the M^{th} pulse is generated at a time allowing interaction with the underlying FATAL algorithm. This requirement translates to being able to make the ratio $(T_2 + T_4)/(\vartheta(T_2 + T_3 + 4d))$ arbitrarily large: $(T_2 + T_4)/\vartheta$ is the minimal gap between successive (FATAL) pulses generated at each node, *provided* that the states of all the NEXT signals are constantly zero (by Inequality (3), T_2 is much larger than T_1 and therefore controls how long it takes to reach *ready* from *accept*), and $T_2 + T_3 + 4d$ is the sum of the maximal desynchronization between synchronized FATAL nodes ($2d$) and the maximal time it takes a node to switch to *ready* and for T_3 to expire after generating a (FATAL) pulse (at this point, nodes will be ready to respond to NEXT_i being true by switching to *propose*).

LEMMA 3.4. *For any $d, \vartheta \in \mathcal{O}(1)$, Condition 3.3 can be satisfied with $T_1, \ldots, T_7, R_1 \in \mathcal{O}(1)$ and $R_2 \in \mathcal{O}(n)$, where the ratio*

$$\alpha := \frac{(T_2 + T_4)/\vartheta}{T_2 + T_3 + 4d}$$

may be chosen to be an arbitrarily large constant.

PROOF. First, observe that if Inequality (3) holds, the denominator in the right hand side of Inequality (12) is positive. Thus, we can equivalently state Inequality (12) as

$$T_2 \geq \frac{2\vartheta\Delta_g + (1 - \lambda)(\vartheta - 1)T_1 + (2 - \lambda)\vartheta d}{1 - \lambda}\ . \tag{13}$$

Since $\lambda \in (4/5, 1)$, this inequality clearly imposes a stronger constraint than Inequality (3), hence we can replace Inequalities (3) and (12) with this one and obtain an equivalent system. The requirement of $(T_2 + T_4)/(\vartheta(T_2 + T_3 + 4d)) = \alpha$ can be rephrased as

$$T_4 \geq (\alpha\vartheta - 1)T_2 + \alpha\vartheta(T_3 + 4d)\ . \tag{14}$$

Again, clearly this constraint is stronger than Inequality (5), hence we drop Inequality (5) in favor of Inequality (14).

We satisfy the inequalities by iteratively defining the values of the left hand sides in accordance with the respective constraint, in the order (2), (13), (7), (4), (14), (6), (8), (9), and finally (10). Note that this is feasible, as in each step the right hand side of the

current inequality is an expression in d, ϑ, α, and, in case of Inequality (10), $n-f$.[24] We obtain the solution

$$
\begin{aligned}
T_1 &:= 4\vartheta d \\
T_2 &:= \frac{50\vartheta^3 d}{1-\lambda} \\
T_6 &:= \frac{50\vartheta^4 d}{1-\lambda} \\
T_3 &:= \frac{(\vartheta^2-1)50\vartheta^3 d}{1-\lambda} + 31\vartheta^3 d \\
T_4 &:= \frac{(\alpha\vartheta^3-1)50\vartheta^3 d}{1-\lambda} + 35\alpha\vartheta^4 d \\
T_5 &:= \frac{(\alpha\vartheta^3-1)50\vartheta^4 d}{1-\lambda} + 39\alpha\vartheta^5 d \\
T_7 &:= \frac{100\alpha\vartheta^8 d}{1-\lambda} + 90\alpha\vartheta^5 d \\
R_1 &:= \frac{(3\alpha\vartheta^3-1)50\vartheta^5 d}{1-\lambda} + 121\alpha\vartheta^6 d \\
R_2 &:= \frac{((3\alpha\vartheta^3-1)100\vartheta^6 + (1-\lambda)(242\alpha\vartheta^7+310\vartheta^2))(n-f)d}{(1-\lambda)^2}\,.
\end{aligned}
$$

As $\alpha \in \mathcal{O}(1)$ was arbitrary, d and ϑ are constants, and $\lambda \in (4/5,1)$ depends on ϑ only and is thus a constant as well, these values satisfy the asymptotic bounds stated in the lemma, concluding the proof. □

4. ANALYSIS

In this section we show that the presented protocol is a (W,E)-stabilizing pulse synchronization protocol, for proper choices of the set of nodes W, the set of channels E, the skew bound Σ, and the accuracy bounds T^-, T^+, stabilizing in time $T(k) \in \mathcal{O}(kn)$ with probability $1-2^{-k(n-f)}$ (for any $k \in \mathbb{N}$). Moreover, we show that if a set of at least $n-f$ nodes fire pulses within the accuracy bounds, then other non-faulty nodes synchronize within $\mathcal{O}(R_1)$ time deterministically.

4.1. Basic Statements

To start our analysis, we need to define the basic requirements for stabilization. Essentially, we need that a large set W of at least $n-f > 2n/3$ nodes is non-faulty and the channels between them are correct. However, the first part of the stabilization process is simply that nodes "forget" about past events that are captured by their timeouts, as the associated information may be incomplete or simply wrong. Therefore, we demand that the nodes in W indeed have been non-faulty for a time period that is sufficiently large to ensure that all timeouts have been reset at least once after the considered set of nodes became non-faulty. The largest possible timeout duration is determined by the maximal possible value of R_3, $\vartheta(R_2+3d)+8(1-\lambda)R_2$. We add another d time to account for the maximum delay, to make sure that these resets in fact took place when the information conveyed by the channels in $W \times W$ has been accurate.

[24] For simplicity, we refrain from demanding equality and drop terms in order to get more condensed expressions. For $\vartheta \leq 1.2$, for example, the increase in the bounds is not significant.

Definition 4.1 (*Coherent Nodes*). The subset of nodes $W \subseteq V$ is called *coherent* during the time interval $[t^-, t^+]$, iff during $[t^- - (\vartheta(R_2 + 3d) + 8(1-\lambda)R_2) - d, t^+]$ all nodes in W are non-faulty, and all channels $S_{i,j}$, $i, j \in W$, are correct.

We will show that if a coherent set of at least $n - f$ nodes fires a pulse, i.e., switches to *accept*, in tight synchrony, this set will henceforth generate pulses deterministically and with controlled frequency, as long as the set remains coherent. This motivates the following definitions.

Definition 4.2 (*Stabilization Points*). We call time t a *W-stabilization point (quasi-stabilization point)* iff all nodes in W switch to *accept* during $[t, t+2d)$ ($[t, t+3d)$).

Throughout this section, we assume the set of coherent nodes W with $|W| \geq n - f$ to be fixed and consider all nodes in and channels originating from $V \setminus W$ as (potentially) faulty.

As a first step, we observe that at times when W is coherent, indeed all nodes reset their timeouts, basing the respective state transitions on proper perception of nodes in W.

LEMMA 4.3. *If W is coherent during the time interval $[t^-, t^+]$, with $t^- \geq \vartheta(R_2+3d)+8(1-\lambda)R_2 + d$, any (randomized) timeout (T, s) of any node $i \in W$ expiring at a time $t \in [t^-, t^+]$ has been reset at least once since time $t^-(\vartheta(R_2+3d)+8(1-\lambda)R_2)$. If t' denotes the time when such a reset occurred, for any $j \in W$ it holds that $S_{i,j}(t') = S_j(\tau_{j,i}^{-1}(t'))$, where $\tau_{j,i}^{-1}(t') \geq t' - d$, i.e., at time t', i observes j in a state j attained when it was non-faulty.*

PROOF. According to Condition 3.3, the largest possible value of any (randomized) timeout is $\vartheta(R_2 + 3d) + 8(1 - \lambda)R_2$. Hence, any timeout that is not expired at time $t^- - (\vartheta(R_2 + 3d) + 8(1 - \lambda)R_2) \geq 0$ expires before time t^- or is reset at least once. As by the definition of coherency all nodes in W are non-faulty and all channels between such nodes are correct during $[t^- - (\vartheta(R_2 + 3d) + 8(1 - \lambda)R_2), t^+]$, this implies the statement of the lemma. □

Phrased informally, any corruption of timeouts and channel states eventually wears off, as correct timeouts expire and correct links remember no events that lie d or more time in the past. Proper cleaning of the memory flags is more complicated and will be explained further down the road.

Throughout this section, we will assume for the sake of simplicity that the set W is coherent at all times and use this lemma implicitly, e.g., we will always assume that nodes from W will observe all other nodes from W in states that they indeed had been at up to less than d time ago, expiring of randomized timeouts at non-faulty nodes cannot be predicted accurately, etc. We will discuss more general settings in Section 5.

We proceed by showing that once all nodes in W switch to *accept* in a short period of time, i.e., a W-quasi-stabilization point is reached, the algorithm guarantees that synchronized pulses are generated deterministically with a frequency that is bounded both from above and below.

THEOREM 4.4. *Suppose t is a W-quasi-stabilization point. Then*

(i) all nodes in W switch to accept *exactly once within $[t, t + 3d)$, and do not leave* accept *until $t + 4d$; and*
(ii) there will be a W-stabilization point $t' \in (t + (T_2 + T_3)/\vartheta, t + T_2 + T_4 + 5d)$ satisfying that no node in W switches to accept *in the time interval $[t + 3d, t')$; and that*
(iii) each node i's, $i \in W$, core state machine (Figure 1) is metastability-free during $[t + 3d, t' + 3d]$.

PROOF OUTLINE. We show that the initial synchrony provided by a W-quasi-stabilization point ensures that all nodes will switch through states *accept*, *sleep*, *sleep* $\rightarrow$ *waking*, *waking*, *ready*, *propose*, *accept* and re-establish tight synchrony by the voting mechanism for switching from *ready* over *propose* to *accept*, as described in Section 3.2 and Figure 2. The time bounds on t' follow from examining the timeouts involved in these transitions, and some additional "reserve" in the timeout constraints guaranteed by Condition 3.3 ensure that each state is safely attained before the next state transition occurs. □

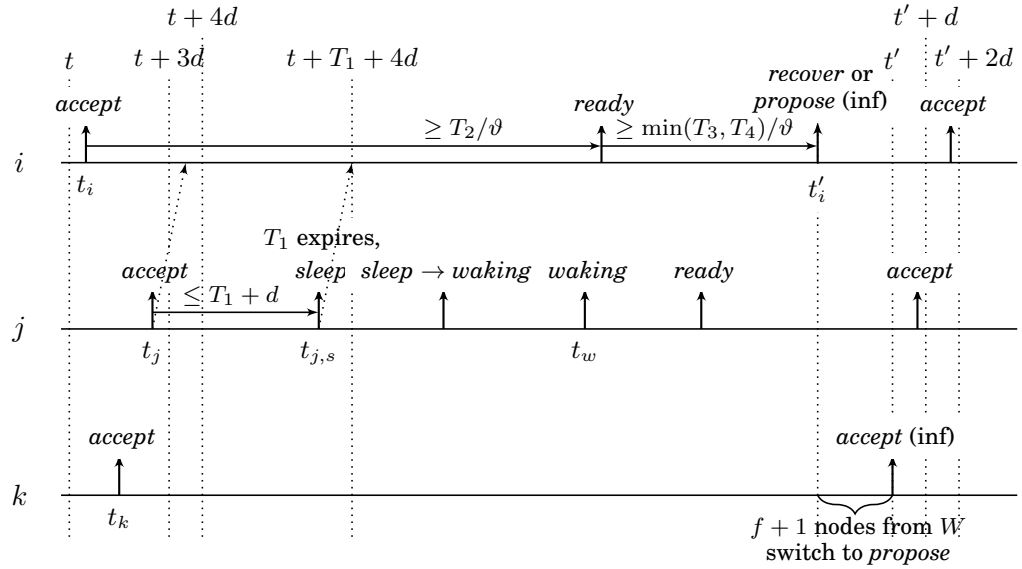

Fig. 10. Part of an execution discussed in the proof of Statement (ii) of Theorem 4.4.

PROOF. Proof of (i): Due to Inequality (2), a node does not leave the state *accept* earlier than $T_1/\vartheta \geq 4d$ time after switching to it. Thus, no node can switch to *accept* twice during $[t, t+3d)$. By definition of a quasi-stabilization point, every node does switch to *accept* in the interval $[t, t+3d) \subset [t, t+T_1/\vartheta)$. This proves Statement (i).

Proof of (ii): For each $i \in W$, let $t_i \in [t, t+3d)$ be the time when i switches to *accept*. By (i) t_i is well-defined. Further let t_i' be the infimum of times in (t_i, ∞) when i switches to *recover* or *propose*.[25] In the following, denote by $i \in W$ a node with minimal t_i'.

Figure 10 depicts part of an execution discussed in the following. We will show that all nodes switch to *propose* via states *sleep*, *sleep* $\rightarrow$ *waking*, *waking*, and *ready* in the presented order. By (i) nodes do not leave *accept* before $t+4d$. Thus at time $t+4d$, each node in W is in state *accept* and observes each other node in W in *accept*. Hence, each node in W memorizes each other node in W in *accept* at time $t+4d$. For each node $j \in W$, let $t_{j,s}$ be the time node j's timeout T_1 expires first after t_j. Then $t_{j,s} \in (t_j + T_1/\vartheta, t_j + T_1 + d)$.[26] Since $|W| \geq n-f$, each node j switches to state *sleep* at time

[25] Note that we follow the convention that $\inf \emptyset = \infty$ if the infimum is taken with respect to a (from above) unbounded subset of $\mathbb{R}_0^+$.

[26] The upper bound comprises an additive term of d since T_1 is reset at some time from $(t_j, t_j + d)$.

$t_{j,s}$. Hence, by time $t + T_1 + 4d$, no node will be observed in state *accept* anymore (until the time when it switches to *accept* again).

When a node $j \in W$ switches to state *waking* at the minimal time t_w larger than t_j, it does not do so earlier than at time $t+T_1/\vartheta+(2+1/\vartheta)T_1 = t+(2+2/\vartheta)T_1 > t+T_1+4d$. As we just observed, this implies that all nodes in W have already left *accept* at least d time ago. Moreover, they cannot switch to *accept* again until t_i' as it is minimal and nodes need to switch to *propose* or *recover* before switching to *accept*. Hence, nodes in W are not observed in state *accept* during $(t + T_1 + 4d, t_i']$, in particular not by node j. Furthermore, by the definition of t_i' and the choice of i, nodes in W are not observed in state *recover* during $(t_w - d, t_i']$. As it resets its *accept* and *recover* flags upon switching to *waking*, j will hence neither switch from *waking* to *recover* nor from *ready* to *propose* during (t_w, t_i').

Now consider node i. By the previous observation, it will not switch from *waking* to *recover*, but to *ready*, following the basic cycle. Consequently, it must wait for timeout T_2 to expire, i.e., cannot switch to *ready* earlier than at time $t + T_2/\vartheta$. By definition of t_i', node i thus switches to *propose* at time t_i'. As it is the first node that does so, and it reset its *propose* flag upon switching to *ready* (at a time larger than t_w, when all nodes from W were already observed in states *sleep*, *sleep* $\rightarrow$ *waking*, *waking*, or *ready*) this cannot happen before timeouts T_3 or T_4 expire, i.e., before time

$$t + \frac{T_2}{\vartheta} + \frac{\min\{T_3, T_4\}}{\vartheta} \overset{(5)}{=} t + \frac{T_2 + T_3}{\vartheta} \overset{(4)}{>} t + T_2 + 5d \,. \tag{15}$$

All other nodes $j \in W$ will switch to *waking* and, for the first time after t_j, observe themselves in state *waking* at a time within $(t + T_1 + 4d, t + (2 + \vartheta)T_1 + 7d)$. Recall that unless they memorize at least $f + 1$ nodes in *accept* or *recover* while being in state *waking*, they will all switch to state *ready* by time

$$\max\{t + T_2 + 4d, t + (2\vartheta + 2)T_1 + 7d\} \overset{(3)}{=} t + T_2 + 4d \,. \tag{16}$$

As we just showed that $t_i' > t + T_2 + 5d$, this implies that at time $t + T_2 + 5d$ all nodes in W are observed in state *ready*, and none of them leaves before time t_i'.

Now choose t' to be the infimum of times from $(t+(T_2+T_3)/\vartheta, t+T_2+T_4+4d]$ (depicted as the shaded area in Figure 10) when a node in W switches to state *accept*.[27] Because of Inequality (15), node j cannot switch to *propose* within $[t_j, t + (T_2 + T_3)/\vartheta)$. Thus, (after time $t + 3d$) node j does not switch to *accept* again earlier than time t', and timeout T_5 cannot expire at j until time

$$t + \frac{T_2 + T_3 + T_5}{\vartheta} \overset{(6)}{\geq} t + T_2 + T_4 + 7d \geq t' + 3d \,, \tag{17}$$

making it impossible for j to switch from *propose* to *recover* at a time within $[t_j, t' + 3d]$. What is more, a node from W that switches to *accept* must stay there for at least $T_1/\vartheta > 3d$ time. Thus, by definition of t', no node $j \in W$ can switch from *accept* to *recover* at a time within $[t_j, t' + 3d]$. Hence, no node $j \in W$ can switch to state *recover* after t_j, but earlier than time $t' + 3d$. It follows that no node in W can switch to other states than *propose* or *accept* during $[t + T_2 + 4d, t' + 3d]$. In particular, no node in W resets its *propose* flags during $[t + T_2 + 5d, t' + 2d] \supset [t_i', t' + 2d]$.

If at time t' a node in W, say node k, switches to state *accept*, $n - 2f \geq f + 1$ of its *propose* flags corresponding to nodes in W are true, i.e., all correspond to a flag in state 1. As the node resets its *propose* flags at the most recent time when it switched to *ready* and no nodes from W have been observed in *propose* between this time and

[27] Note that since we take the infimum on $(t+(T_2+T_3)/\vartheta, t+T_2+T_4+4d]$, we have that $t' \leq t+T_2+T_4+4d$.

t_i', it holds that $f+1$ nodes in W switched to state *propose* during $[t_i', t')$. Since we established that no node resets its *propose* flags during $[t_i', t'+2d]$, it follows that all nodes are in state *propose* by time $t'+d$. Consequently, all nodes in W will observe all nodes in W in state *propose* before time $t'+2d$ and switch to *accept*, i.e., $t' \in (t+(T_2+T_3)/\vartheta, t+T_2+T_4+4d)$ is a W-stabilization point. Statement (ii) follows.

On the other hand, if at time t' no node in W switches to state *accept*, it follows that $t' = t+T_2+T_4+4d$, by definition of the infimum. As all nodes observe themselves in state *ready* by time $t+T_2+5d$, they switch to *propose* before time $t+T_2+T_4+5d = t'+d$ because T_4 expired. By the same reasoning as in the previous case, they switch to *accept* before time $t'+2d$, i.e., Statement (ii) holds as well.

Proof of (iii): We have shown that within $[t_j, t'+3d]$, any node $j \in W$ switches to states along the basic cycle only. To show the correctness of Statement (iii), it is thus sufficient to prove that, whenever j switches from state s of the basic cycle to s' of the basic cycle during time $[t_j, t'+3d]$, the transition from s to *recover* is disabled from the time it switches to s' until it observes itself in this state. We consider transitions $tr(\textit{accept}, \textit{recover})$, $tr(\textit{waking}, \textit{recover})$, and $tr(\textit{propose}, \textit{recover})$ one after the other:

(1) $tr(\textit{accept}, \textit{recover})$: We showed that node j's $tr(\textit{accept}, \textit{sleep})$ is satisfied before time $t+4d \le t+T_1/\vartheta$, i.e., before $tr(\textit{accept}, \textit{recover})$ can hold, and no node resets its *accept* flags less than d time after switching to state *sleep*. When j switches to state *accept* again at or after time t', T_1 will not expire earlier than time $t'+4d$.
(2) $tr(\textit{waking}, \textit{recover})$: As part of the reasoning about Statement (ii), we derived that $tr(\textit{waking}, \textit{recover})$ does not hold at nodes from W observing themselves in state *waking*.
(3) $tr(\textit{propose}, \textit{recover})$: The additional slack of d in Inequality (17) ensures that T_5 does not expire at any node in W switching to state *accept* during $(t', t'+2d)$ earlier than time $t'+3d$.

Since $[t_j, t'+3d] \supset [t+3d, t'+3d]$, Statement (iii) follows. □

Inductive application of Theorem 4.4 shows that by construction of our algorithm, nodes in W provably do not suffer from metastability upsets once a W-quasi-stabilization point is reached, as long as all nodes in W remain non-faulty and the channels connecting them are correct. Unfortunately, it can be shown that it is impossible to ensure this property during the stabilization period, thus rendering a formal treatment infeasible. This is not a peculiarity of our system model, but a threat to any model that allows for the possibility of metastable upsets as encountered in physical chip designs. However, it was shown that, by proper chip design, the probability of metastable upsets can be made arbitrarily small [Fuchs et al. 2009].[28] For the purpose of the stabilization analysis in the remainder of this section and Section 5, *we will therefore assume that all non-faulty nodes are metastability-free in all executions.*

The next lemma reveals a very basic property of the main algorithm that is satisfied if no non-faulty nodes may switch to state *join* in a given period of time. It states that if a non-faulty node switches to state *sleep*, other non-faulty nodes cannot remain too far ahead or behind in case they are on the basic cycle as well.

[28]Note that it is feasible to incorporate this issue into the model by means of the probability space, as it is beyond control of "reasonable" adversaries to control signals on faulty channels (or ones that originate at non-faulty nodes) precisely enough to ensure more than a small probability of a metastable upset. However, since it is (at best) impractical to consider metastable states of the system in our theoretical framework, essentially this approach boils down to counting the number of state transitions during stabilization where a non-faulty node is in danger of becoming metastable and control this risk by means of the union bound.

LEMMA 4.5. *Assume that at time* t_{sleep}*, some node from* W *switches to* sleep *and no node from* W *is in state* join *during* $[t_{\text{sleep}} - T_1 - 2d, t_{\text{sleep}} + 2T_1 + 2d]$*. Then*

(i) at time $t_{\text{sleep}} + 2T_1 + 2d$*, each node in* W *is in one of the states* sleep*,* sleep $\rightarrow$ waking*,* waking*, or* recover*;*

(ii) each node from W *in states* sleep*,* sleep $\rightarrow$ waking*, or* waking *at time* $t_{\text{sleep}} + 2T_1 + 2d$ *switches to* accept *and resets its timeout* (T_2, accept) *(most recently w.r.t. time* $t_{\text{sleep}} + 2T_1 + 2d$*) at times from* $(t_{\text{sleep}} - \Delta_g - 4d, t_{\text{sleep}} + T_1 + 2d)$*; and*

(iii) no node in W *switches from* recover *to* accept *during* $[t_{\text{sleep}} + T_1 + d, t_a]$*, where* $t_a > t_{\text{sleep}} + 2T_1 + 2d$ *denotes the infimum of times larger than* $t_{\text{sleep}} + T_1 + d$ *when a node in* W *switches to state* accept*.*

PROOF OUTLINE. For the sake of the argument, assume first that no node in W switches from *recover* to *accept*. For switching to *sleep*, a node in W must observe at least $n-f$ nodes in *accept* within the fairly short time of roughly T_1 it stays in *accept*, at least $n-2f$ of which are non-faulty. Once these nodes leave *accept*, either to *sleep* or to *recover*, the only way to get to *accept* again is by following the basic cycle (as switching to *join* and the transition from *recover* to *accept* are forbidden), which requires at the least T_2 to expire. Without these $n-2f$ nodes in *accept*, no other node in W can switch to *sleep*, as $2(n-2f) > n-f = |W|$, the number of non-faulty nodes available. From this we can conclude that at time $t_{sleep} + 2T_1 + 2d$ all nodes in W are roughly in the same spot of the basic cycle or in *recover* (Statements (i) and (ii)).

Regarding the possible transition from *recover* to *accept*, we argue that at least $n-2f$ nodes from W that enabled the original transition of some node in W to *sleep* at time t_{sleep} cannot be (observed) in state *accept* at time $t_{sleep} + T_1 + d$, as after leaving *accept* to either *recover* or *sleep* sufficiently long timeouts need to expire first. However, this also entails that the transition from *recover* to *accept* becomes impossible starting on time $t_{sleep} + T_1 + d$ until some of these at least $n-2f$ nodes reach *accept* again following the basic cycle (or via *join*). This implies Statement (iii) and shows Statements (i) and (ii) in the general case. □

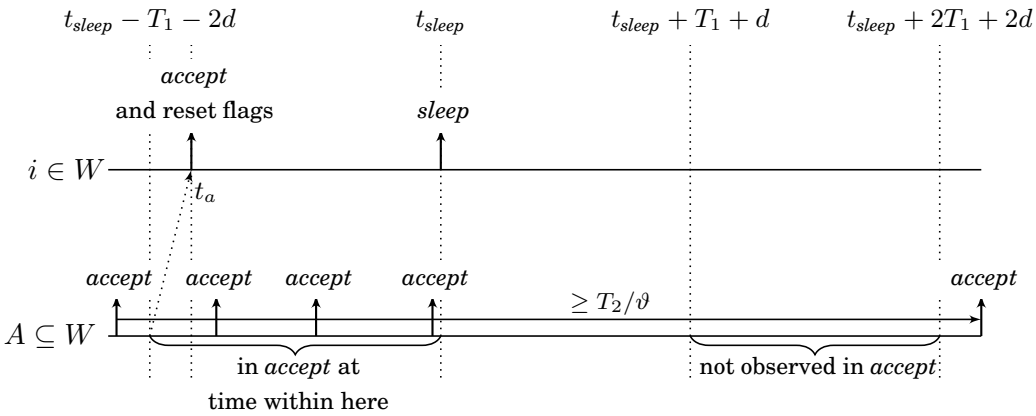

Fig. 11. Part of an execution discussed in the proof of Statement (iii) of Lemma 4.5.

PROOF. We will first show that Statement (iii) holds. Figure 11 depicts a part of the execution discussed in the following. We claim that there is a subset $A \subseteq W$ of at least $n-2f$ nodes such that each node from A has been in state *accept* at some time

in the interval $(t_{sleep} - T_1 - 2d, t_{sleep})$. To see this, observe that if a node in W (node i in Figure 11) switches to state *sleep* at time t_{sleep}, it must have observed $n - 2f$ non-faulty nodes from W in state *accept* at times from $(t_{sleep} - T_1 - d, t_{sleep}]$, since it resets its *accept* flags after switching to *accept* at the time $t_a \geq t_{sleep} - T_1 - d$ (that is minimal with this property). Each of these nodes must have been in state *accept* at some time from $(t_{sleep} - T_1 - 2d, t_{sleep})$, showing the existence of a set $A \subseteq W$ as claimed.

At time

$$t_{sleep} + T_1 + d \overset{(3)}{=} t_{sleep} - T_1 - 2d + \min\left\{\frac{\vartheta(2T_1 + 3d)}{\vartheta}, \frac{T_2}{\vartheta} - (T_1 + d)\right\} ,$$

no node from A is observed in state *accept* by the following arguments. First, it must have left *accept* by $t_{sleep} + T_1$. Second, we claim that it cannot have switched to *accept* again by that time. To see this, observe that, assuming otherwise, it must have switched to *accept* again at a time greater than $t_{sleep} - T_1 - 2d - (T_1 + d)$. By assumption, it cannot have switched to *accept* via *join*, and in order to switch directly from *recover* to *accept*, a timeout of $\vartheta(2T_1 + 3d)$ needs to expire first. Since this in particular applies to the nodes from A and no node from W is in state *join* until time $t_{sleep} + 2T_1 + 2d$, the only way to switch to *accept* is thus by following the basic cycle via states *sleep*, *sleep* $\rightarrow$ *waking*, *waking*, *ready*, and *propose*. However, this takes at least until time

$$t_{sleep} - T_1 - 2d - (T_1 + d) + \frac{T_2}{\vartheta} \overset{(3)}{>} t_{sleep} + 2T_1 + 2d$$

as well. Thus, the claim holds and no node in W observes a node in A in *accept* within time $[t_{sleep} + T_1 + d, t_{sleep} + 2T_1 + 2d]$. Since $|A| \geq n - 2f \geq f + 1$, this implies that no node in W can switch from *recover* to *accept* without observing some node from A in *accept*. Statement (iii) of the lemma follows.

We next show Statements (i) and (ii) for any node in W (node i in Figure 12) that observes itself in one of the states *waking*, *ready*, or *propose* at time $t_{sleep} - T_1 - 2d$. By time $t_{sleep} + d$, it will memorize all nodes from A in *accept* (provided that it did not switch to *accept* in the meantime). Hence, it satisfies $tr(\textit{waking}, \textit{recover})$, $tr(\textit{ready}, \textit{propose})$, and $tr(\textit{propose}, \textit{accept})$ until it switches to either *recover* or *accept*. It follows that any such node must have switched to *recover* or *accept* by time $t_{sleep} + 3d < t_{sleep} + T_1 + d$. Since the transition from *recover* to *join* is excluded by the assumptions of the lemma, either (a) the node switches to *sleep* at some time during the interval $(t_{sleep} - T_1 - 2d, t_{sleep} + 2T_1 + 2d)$, or (b) it will be in state *accept* or *recover* at time $t_{sleep} + T_1 + d$. In case it is in *accept*, it will switch to *recover* by time $t_{sleep} + 2T_1 + 2d$, after its timeout T_1 expired. Since nodes in W cannot switch from *recover* to *accept* during $[t_{sleep} + T_1 + d, t_a)$, with $t_a > t_{sleep} + 2T_1 + 2d$, because of Statement (iii), it follows that, in case of (b), it will be in state *recover* by time $t_{sleep} + 2T_1 + 2d$, without the possibility to leave earlier than time t_a. Thus Statements (i) and (ii) follow for case (b).

In case of (a), if the node switched to *sleep* during $(t_{sleep} - T_1 - 2d, t_{sleep} + 2T_1 + 2d)$, consider the most recent time before that when it switched to *accept*. We know that this must have happened during $(t_{sleep} - T_1 - 3d, t_{sleep} + T_1 + d)$; the lower bound follows from the fact that it observes itself in *waking*, *ready* or *propose* at time $t_{sleep} - T_1 - 2d$, and thus must have switched to *accept* after time $t_{sleep} - T_1 - 3d$. The upper bound follows from the observations before. And since the only way to reach *sleep* from these states is over state *accept*, we know that such a time indeed exists. Thus the node must have reset its timeout $(T_2, \textit{accept})$ at some time from $(t_{sleep} - T_1 - 3d, t_{sleep} + T_1 + 2d)$. Hence Statement (ii) is satisfied for nodes observing themselves in states *waking*, *ready*, or *propose* at time $t_{sleep} - T_1 - 2d$. Considering that

$$t_{sleep} - (2\vartheta + 3)T_1 - 6d + \frac{T_2}{\vartheta} \overset{(3)}{>} t_{sleep} + 2T_1 + 2d , \tag{18}$$

Statement (i) follows for such nodes, as T_2 cannot be expired by time $t_{sleep} + 2T_1 + 2d$.

Note that we can reason analogously for nodes that already observe themselves in states *recover* or *accept* at time $t_{sleep} - T_1 - 2d$; the only difference is that such a node may already have switched to *accept* up to $T_1 + d$ time earlier and thus resets its timeout T_2 at some time from $(t_{sleep} - 2T_1 - 4d, t_{sleep} + T_1 + d)$, which is well in the bounds claimed by Statement (ii). By Inequality (18), Statement (i) holds as well.

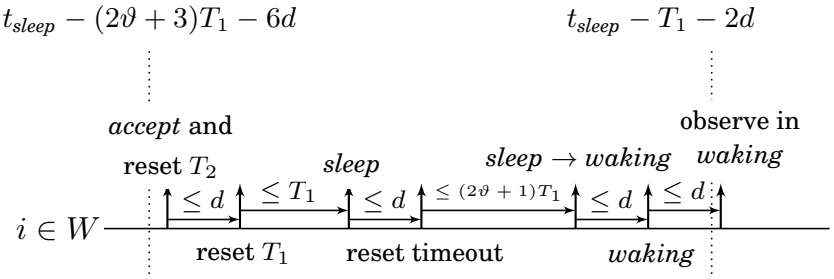

Fig. 12. Part of an execution discussed in the proof of Statements (i) and (ii) of Lemma 4.5.

Hence, it remains to consider nodes that observe themselves in states *sleep* or *sleep* → *waking* at time $t_{sleep} - T_1 - 2d$. Such a node must have switched to *sleep* at a time from $(t_{sleep} - (2\vartheta + 2)T_1 - 5d, t_{sleep} - T_1 - 2d)$, cf. Figure 12. Each considered node thus switched to *accept* (most recently before this time) at some time from $(t_{sleep} - (2\vartheta + 3)T_1 - 6d, t_{sleep} - T_1 - 2d)$, and reset its timeout T_2 most recently at a time from $(t_{sleep} - (2\vartheta + 3)T_1 - 6d, t_{sleep} - T_1 - d) = (t_{sleep} - \Delta_g - 4d, t_{sleep} - T_1 - d)$, yielding Statement (ii) for the respective node. Applying Inequality (18) once more, we obtain Statement (i). As we have shown Statements (i) and (ii) for all possible states of the main state machine a node in W can observe itself in at time $t_{sleep} - T_1 - 2d$, this concludes the proof. □

Granted that nodes are not in state *join* for sufficiently long, this implies that nodes will switch to *sleep* in rough synchrony with others or drop out of the basic cycle and switch to *recover*.

COROLLARY 4.6. *Assume that at time* t_{sleep}*, a node from* W *switches to* sleep*, no node is in state* join *during* $[t_{\text{sleep}} - T_1 - 2d, t_{\text{sleep}} + 2T_1 + 3d]$*, and also that during* $(t_{\text{sleep}} - \Delta_g, t_{\text{sleep}}) = (t_{\text{sleep}} - (2\vartheta + 3)T_1 - 2d, t_{\text{sleep}})$ *no node in* W *is in state* sleep*. Then at time* $t_{\text{sleep}} + 2T_1 + 3d$*, any node* i *from* W *is either in one of the states* sleep *or* sleep → waking *and observed in* sleep*, or it is and is observed in state* recover*.*

PROOF. For a node $i \in W$, denote by t_i the supremum of times from $[0, t_{sleep} + 2T_1 + 3d]$ when the node switches to state *sleep*. We distinguish four cases.

(1) $t_i \in [t_{sleep} + 2T_1 + 2d, t_{sleep} + 2T_1 + 3d]$: By Statement (i) of Lemma 4.5, node i is not in state *accept* at time $t_{sleep} + 2T_1 + 2d$, and hence cannot switch to *sleep* until time $t_i + T_1/\vartheta > t_{sleep} + 2T_1 + 3d$, a contradiction. Thus this does not apply.

(2) $t_i \in [t_{sleep}, t_{sleep} + 2T_1 + 2d]$: In this case, the node will stay in state *sleep* at least until time $t_i + (2\vartheta + 1)T_1/\vartheta > t_{sleep} + 2T_1 + 3d$ (by Inequality (2)).

(3) $t_i \in [t_{sleep} - \Delta_g, t_{sleep}]$: This does not apply by assumption.

(4) $t_i < t_{sleep} - \Delta_g$: By Statements (i) and (ii) of Lemma 4.5, all nodes $j \in W$ are either in state *recover* at time $t_{sleep} + 2T_1 + 2d$ or switched to *accept* at a time from $(t_{sleep} - \Delta_g - 4d, \min\{t_j, t_{sleep} + T_1 + 2d\}) \subseteq (t_{sleep} - \Delta_g - T_1/\vartheta, t_{sleep} + T_1 + 2d)$. We will exclude the latter case. To this end, recall that any non-faulty node j will stay in

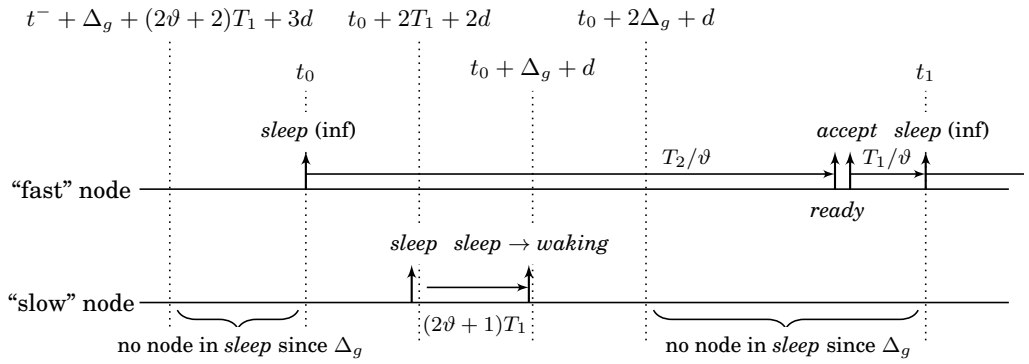

Fig. 13. Part of an execution as considered in the induction in Lemma 4.7.

state *accept* for at least T_1/ϑ time before switching to *sleep*. Hence, we conclude that if $t_j < t_{sleep} - \Delta_g$, the most recent time when it switched to *accept* before t_j is smaller than $t_{sleep} - \Delta_g - T_1/\vartheta$, a contradiction. Hence, indeed we have that any non-faulty node j is in state *recover* at time $t_{sleep} + 2T_1 + 2d$ (which in particular holds for $j = i$).
By Statement (iii) of Lemma 4.5, no node from W switches from *recover* to *accept* during $[t_{sleep} + 2T_1 + 2d, \min\{t_a, t_{sleep} + 2T_1 + 3d\}]$, where t_a is the infimal time larger than $t_{sleep} + T_1 + d$ when a node from W switches to *accept*. Nodes from the first three cases switch to *accept* (again) after time $t_{sleep} + 2T_1 + 3d$, because of the timeouts along the basic cycle. Because no node from W switches to *join* during $[t_{sleep} + 2T_1 + 2d, t_{sleep} + 2T_1 + 3d]$, also nodes to which the current case applies switch to *accept* after time $t_{sleep} + 2T_1 + 3d$. Hence, $\min\{t_a, t_{sleep} + 2T_1 + 3d\} = t_{sleep} + 2T_1 + 3d$, and we conclude that node i is in state *recover* during $[t_{sleep} + 2T_1 + 2d, t_{sleep} + 2T_1 + 3d]$ for this case.

Hence, node i satisfies the claim in all cases, concluding the proof. □

4.2. Resynchronization Points

In this section, we derive that within linear time, it is very likely that good resynchronization points occur. As a first step, we infer from Lemma 4.5 that for any time interval during which no node from W enters state *join*, most of the time in this interval no nodes from W are in state *sleep*.

LEMMA 4.7. *Suppose no node in W is in state* join *during $[t^-, t^+]$. Then the volume of times $t \in [t^- + T_1 + 2d, t^+]$ satisfying that no node from W is in state* sleep *during $(t - \Delta_g, t)$ is at least*

$$\left(\frac{T_2 - 2\vartheta\Delta_g - (\vartheta-1)T_1 - 2\vartheta d}{T_2 - (\vartheta-1)T_1 - \vartheta d}\right)(t^+ - t^-) - 6\Delta_g \ .$$

PROOF OUTLINE. Lemma 4.5 and Corollary 4.6 essentially state that if no node from W may switch to *join*, there will be only one roughly synchronized group of non-faulty nodes executing the basic cycle, while all other non-faulty nodes end up in state *recover*. Moreover, this group cannot switch to *sleep* too frequently, as every time this happens, T_2 must expire at the respective nodes (cf. Figure 13). By choosing T_2 a constant factor larger than T_1 (recall that non-faulty nodes will leave *sleep* after a timeout of $(2\vartheta+1)T_1$ expires), we can thus guarantee that a large fraction of the time no non-faulty node is in state *sleep*. We prove this by induction on the times when (the first node of) the group executing the basic cycle switches to state *sleep*. Naturally, it may also happen that all non-faulty nodes cease executing the basic cycle. This just makes

things even simpler, though. Once the last non-faulty node leaves *sleep*, no non-faulty node will be in state *sleep* again until time t^+. □

PROOF. Denote by t_0 the infimum of times from $[t^- + T_1 + 2d, t^+]$ when a node from W switches to *sleep*. Recall that any such node will leave state *sleep* once a timeout of $(2\vartheta + 1)T_1$ expires, that is, at most $(2\vartheta + 1)T_1 + d$ time after switching to the state. Thus, any time $t \in [t^- + \Delta_g + (2\vartheta + 2)T_1 + 3d, t_0]$ satisfies that no node in W is in state *sleep* during $(t - \Delta_g, t)$. We proceed by induction over increasing times $t_i \in (t_0, t^+]$, $i \in \{1, \dots, i_{\max}\}$, where t_i is the infimum of times from $[t_{i-1} + 2T_1 + 3d, t^+]$ when a node from W switches to *sleep*. The induction halts at index $i_{\max} \in \mathbb{N}$ iff $t_{i_{\max}} > t^+ - T_2/\vartheta + (1 - 1/\vartheta)T_1 + d$. We claim that, for each i, the volume of the set of times $t \in [t^- + \Delta_g + (2\vartheta + 2)T_1 + 3d, t_i]$ such that no node in W is in state *sleep* during $(t - \Delta_g, t)$ is at least

$$t_i - t^- - (\Delta_g + (2\vartheta + 2)T_1 + 3d) - i(2\Delta_g + d) \tag{19}$$

and that

$$t_i \geq t^- - \left(2\vartheta + 1 + \frac{1}{\vartheta}\right) T_1 - d + i\left(\frac{T_2}{\vartheta} - \left(1 - \frac{1}{\vartheta}\right) T_1 - d\right) . \tag{20}$$

In fact, we will show these bounds by establishing that no node from W is in state *sleep* during

$$(t_{i-1} + (2\vartheta + 3)T_1 + 3d, t_i) = (t_{i-1} + \Delta_g + d, t_i) , \tag{21}$$

that

$$t_i \geq t_{i-1} + \frac{T_2}{\vartheta} - \Delta_g - 4d \overset{(3)}{\geq} t_{i-1} + 2\Delta_g + 3d \tag{22}$$

for all $i \in \{1, \dots, i_{\max}\}$, and proving a slightly stronger version of the latter inequality for indices $i > 1$.

We first establish these bounds for t_1 (cf. Figure 13). By Lemma 4.5, every node in W that is not in state *recover* at time $t_0 + 2T_1 + 2d$ resets T_2 at some time from $(t_0 - \Delta_g - 4d, t_0 + T_1 + 2d)$ and is in one of the states *sleep*, *sleep* $\to$ *waking*, or *waking* at time $t_0 + 2T_1 + 2d$. Hence, such nodes do not switch to state *ready* and subsequently to *propose*, *accept*, and *sleep* again until $t_0 + T_2/\vartheta - \Delta_g - 4d \leq t^+$, i.e., Inequality (22) holds for $i = 1$. We obtain

$$\begin{aligned} t_1 &\geq t_0 + \frac{T_2}{\vartheta} - \Delta_g - 4d \\ &\geq t^- + \frac{T_2}{\vartheta} - \Delta_g + T_1 - 2d \\ &= t^- - \left(2\vartheta + 1 + \frac{1}{\vartheta}\right) T_1 - d + \left(\frac{T_2}{\vartheta} - \left(1 - \frac{1}{\vartheta}\right) T_1 - d\right) , \end{aligned}$$

showing Inequality (20) for $i = 1$. Moreover, Statement (i) of Lemma 4.5 implies that no node in W is in state *sleep* during $[t_0 + (2\vartheta + 3)T_1 + 3d, t_1) = [t_0 + \Delta_g + d, t_1)$, as any node in W in state *sleep* at time $t_0 + 2T_1 + 2d$ will leave after a timeout of $(2\vartheta + 1)T_1$ expires; this shows (21) for $i = 1$. Hence, the volume of the set of times $t \in [t^- + \Delta_g + (2\vartheta + 2)T_1 + 3d, t_1]$ such that no node from W is in state *sleep* during $(t - \Delta_g, t)$ is at least (cf. Figure 13)

$$t_0 - (t^- + \Delta_g + (2\vartheta + 2)T_1 + 3d) + t_1 - (t_0 + 2\Delta_g + d) ,$$

showing Inequality (19) for $i = 1$.

We now perform the induction step from $i < i_{\max}$ to $i + 1$. By (21) for index i, no node from W is in state *sleep* during

$$(t_{i-1} + \Delta_g + d, t_i) \overset{(22)}{\supseteq} (t_i - \Delta_g, t_i) .$$

Hence we can apply Corollary 4.6 to see that nodes in W not observing themselves in state *sleep* at time t_i+2T_1+3d switched to state *recover* (and observe themselves in this state). Therefore, non-faulty nodes that continue to execute the basic cycle must have switched to *sleep* by time $t_i + 2T_1 + 3d$ and have performed their most recent reset of timeout T_2 at or after time t_i-T_1-d. Observe that switching to *sleep* again also requires T_1 to expire after switching to *accept*. Thus, such nodes do not switch to state *ready* and subsequently to *propose*, *accept*, and *sleep* again until time $t_i+T_2/\vartheta-(1-1/\vartheta)T_1-d \leq t^+$ (as otherwise we had $i = i_{\max}$), yielding

$$t_{i+1} \geq t_i + \frac{T_2}{\vartheta} - \left(1 - \frac{1}{\vartheta}\right) T_1 - d \, .$$

Moreover, like for $i = 1$, no node in W is in state *sleep* during $[t_i + \Delta_g + d, t_{i+1})$. These two statements show Inequality (21) and Inequality (22) for $i + 1$, and by means of the induction hypothesis directly imply Inequality (19) and Inequality (20) for $i+1$ as well, i.e., the induction succeeds.

From Inequality (20), we have that

$$i_{\max} \leq \frac{t^+ - t^- + (2\vartheta + 1 + 1/\vartheta)T_1 + d}{T_2/\vartheta - (1 - 1/\vartheta)T_1 - d} \overset{(3)}{<} \frac{t^+ - t^-}{T_2/\vartheta - (1 - 1/\vartheta)T_1 - d} + 1 \, . \tag{23}$$

Observe that the same reasoning as above shows that no node from W switches to *sleep* during $[t_{i_{\max}}+\Delta_g+d, t^+]$ since $t_{i_{\max}} \geq t^+-(T_2/\vartheta-(1-1/\vartheta)T_1-d)$. Thus, inserting $i = i_{\max}$ into Inequality (19), we infer that the volume of times $t \in [t^-+\Delta_g+(2\vartheta+2)T_1+3d, t^+] \subset [t^-, t^+]$ such that no node from W is in state *sleep* during $(t, t - \Delta_g)$ is at least

$$\begin{aligned}
&t^+ - t^- - (\Delta_g + (2\vartheta + 2)T_1 + 3d) - (i_{\max} + 1)(2\Delta_g + d) \\
&\overset{(23)}{>} \left(\frac{T_2 - 2\vartheta\Delta_g - (\vartheta - 1)T_1 - 2\vartheta d}{T_2 - (\vartheta - 1)T_1 - \vartheta d}\right)(t^+ - t^-) - (5\Delta_g + (2\vartheta + 2)T_1 + 5d) \\
&\overset{(2)}{>} \left(\frac{T_2 - 2\vartheta\Delta_g - (\vartheta - 1)T_1 - 2\vartheta d}{T_2 - (\vartheta - 1)T_1 - \vartheta d}\right)(t^+ - t^-) - 6\Delta_g \, ,
\end{aligned}$$

concluding the proof. □

We are now ready to advance to proving that good resynchronization points are likely to occur within bounded time, no matter what the strategy of the adversary is. To this end, we first establish that in any execution, at most of the times a non-faulty node switching to state *init* will result in a good resynchronization point. This is formalized by the following definition.

Definition 4.8 (*Good Times*). Given an execution $\mathcal{E}$ of the system, denote by $\mathcal{E}'$ any execution satisfying that $\mathcal{E}'|_{[0,t)} = \mathcal{E}|_{[0,t)}$, where at time t a node $i \in W$ switches to state *init* in $\mathcal{E}'$. Time t is *good in $\mathcal{E}$ (with respect to W)* provided that for any such $\mathcal{E}'$ it holds that t is a good W-resynchronization point in $\mathcal{E}'$.

The previous statement thus boils down to showing that in any execution, the majority of the times is good.

LEMMA 4.9. *Given any execution $\mathcal{E}$ and any time interval $[t^-, t^+]$, the volume of good times in $\mathcal{E}$ during $[t^-, t^+]$ is at least*

$$\lambda^2(t^+ - t^-) - \frac{11(1 - \lambda)R_2}{10\vartheta} \, .$$

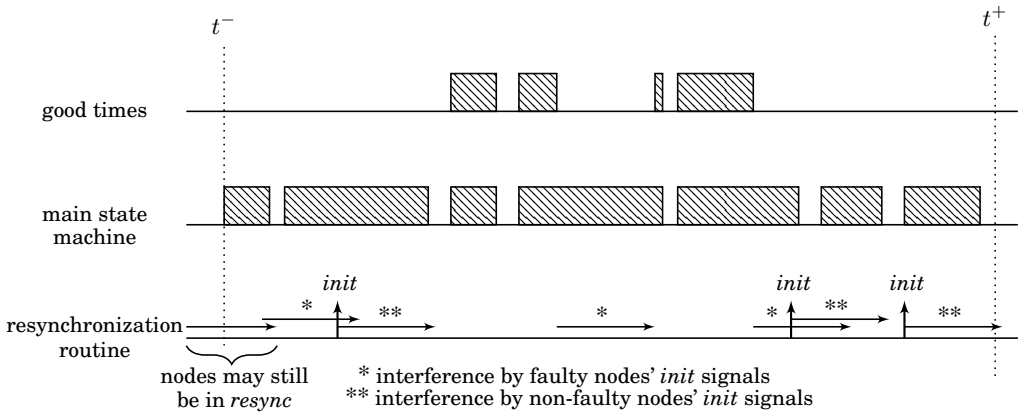

Fig. 14. Illustration of the lower bound on the volume of good times shown in Lemma 4.9. The adversary utilizes the approach shown in Figure 8 to "block" the time ranges marked by *. The *init* signals of non-faulty nodes may also cause interference; the respective ranges are marked by **. Shaded rectangles at the "main state machine" axis mark the times t for which during $(t - \Delta_g, t)$ no node from W is in *sleep* (cf. Lemma 4.7 and Figure 13). Shaded rectangles at the "good times" axis mark good times.

PROOF OUTLINE. The proof goes by discarding a sufficiently large set of times to make sure that all potential reasons for the remaining times not being good are excluded (see Figure 14). As argued in Section 3.5 and shown in Figure 7, we can be sure that a resynchronization point will occur when some non-faulty node switches to *init*, granted that no non-faulty nodes are in either of the states *supp* $\rightarrow$ *resync* or *resync*. As non-faulty nodes will not switch to or remain in state *join* when in *dormant*, and non-faulty nodes will switch to *dormant* when not in *resync*, for most of the times when the first condition holds, we also have that no node from W is in state *join*. From Lemma 4.7 we see that for a large fraction of the remaining times, also no node from W is in state *sleep*. Hence, when at some non-faulty node the roughly uniformly distributed timeout R_3 expires, there is a constant probability that this will result in a good resynchronization point, i.e., a large fraction of the times are good.

This line of reasoning requires one more argument, though. When non-faulty nodes switch to *resync*, they remain there until R_1 expires. Meanwhile, a good resynchronization point cannot occur. Consequently, we need to argue that such transitions do not happen too frequently. We show that—due to the voting scheme employed in the resynchronization routine—non-faulty nodes will switch to *supp* $\rightarrow$ *resync* and *resync* only if a linear number of R_2 timeouts is reset in a small time window. Hence, in $\mathcal{O}(R_2) = \mathcal{O}(R_3)$ time there can be no more than $\mathcal{O}(n)$ such time windows. The timeout constraints in Condition 3.3 make sure that the constants play out in our favor, so that for any time interval that is larger than roughly R_2, a constant fraction of the times must be good. □

PROOF. Assume w.l.o.g. that $|W| = n - f$ (otherwise consider a subset of size $n - f$) and abbreviate

$$\begin{aligned} N &:= \left(\frac{\vartheta(t^+ - t^-)}{R_2} + \frac{11}{10}\right)(n-f) \\ &\geq \left\lceil \frac{\vartheta(t^+ - t^-) + R_2/10}{R_2} \right\rceil (n-f) \\ &\overset{(10)}{\geq} \left\lceil \frac{\vartheta(t^+ - t^-) + \vartheta(R_1 + 6\Delta_g + T_1 + 5d)/(5(1-\lambda))}{R_2} \right\rceil (n-f) \\ &\overset{(1)}{\geq} \left\lceil \frac{\vartheta(t^+ - t^- + R_1 + 6\Delta_g + T_1 + 5d)}{R_2} \right\rceil (n-f)\,. \end{aligned} \tag{24}$$

The proof is in two steps: First we construct a measurable subset of $[t^-, t^+]$ that comprises good times only. In a second step a lower bound on the volume of this set is derived.

Constructing the set: Consider an arbitrary time $t \in [t^-, t^+]$, and assume a node $i \in W$ switches to state *init* at time t. When it does so, its timeout R_3 expires. Recall that W is coherent during $[t^-, t^+]$. Hence, by Lemma 4.3, all timeouts of node i that expire at times within $[t^-, t^+]$ have been reset at least once between the time when the node became non-faulty and t. Let t_{R3} be the maximum time not later than t when R_3 was reset. Due to the distribution of R_3 we know that

$$t_{R3} \overset{(11)}{\leq} t - (R_2 + 3d)\,.$$

Thus, node i is not in state *init* during time $[t - (R_2 + 2d), t)$, and no node $j \in W$ observes i in state *init* during time $[t - (R_2 + d), t)$. Thereby any node j's, $j \in W$, timeout $(R_2, \textit{supp}\ i)$ corresponding to node i is expired at time t.

We claim that the condition that no node from W is in or observed in one of the states *resync* or *supp* $\to$ *resync* at time t is sufficient for t being a W-resynchronization point. To see this, assume that the condition is satisfied. Thus all nodes $j \in W$ are in states *none* or *supp* k for some $k \in \{1, \dots, n\}$ at time t. By the algorithm, they all will switch to state *supp* i or state *supp* $\to$ *resync* during $(t, t+d)$. It might happen that they subsequently switch to another state *supp* k' for some $k' \in V$, but all of them will be in one of the states with signal *supp* during $(t+d, t+2d]$. Consequently, all nodes from W will observe at least $n-f$ nodes in state *supp* during $(t', t+2d)$ for some time $t' < t+2d$. Hence, those nodes in W that were still in state *supp* i (or *supp* k' for some k') at time $t + d$ switch to state *supp* $\to$ *resync* before time $t + 2d$, i.e., t is a W-resynchronization point.

We proceed by analyzing under which conditions t is a good W-resynchronization point. Recall that in order for t to be good, it has to hold that no node from W switches to state *sleep* during $(t - \Delta_g, t)$ or is in state *join* during $(t - T_1 - 2d, t + 4d)$.

We begin by characterizing subsets of good times within $(t_r, t_r') \subset [t^-, t^+]$, where t_r and t_r' are times such that during (t_r, t_r') no node from W switches to state *supp* $\to$ *resync*. Due to timeout

$$R_1 \overset{(9)}{>} (4\vartheta + 2)d\,,$$

we know that during $(t_r + R_1 + 2d, t_r')$, no node from W will be in, or be observed in, states *supp* $\to$ *resync* or *resync*. Thus, if a node from W switches to *init* at a time within $(t_r + R_1 + 2d, t_r')$, it is a W-resynchronization point. Further, all nodes in W will be in

state *dormant* during $(t_r + R_1 + 2d, t'_r + 4d)$. Thus all nodes in W will be observed to be in state *dormant* during $(t_r + R_1 + 3d, t'_r + 4d)$, implying that they are not in state *join* during $(t_r + R_1 + 3d, t'_r + 4d)$. In particular, any time $t \in (t_r + R_1 + T_1 + 5d, t'_r)$ satisfies that no node in W is in state *join* during $(t - T_1 - 2d, t + 4d)$. Applying Lemma 4.7, we infer that the total volume of times from (t_r, t'_r) that is good is at least

$$\left(\frac{T_2 - 2\vartheta\Delta_g - (\vartheta - 1)T_1 - 2\vartheta d}{T_2 - (\vartheta - 1)T_1 - \vartheta d}\right)(t'_r - t_r) - (R_1 + 6\Delta_g + T_1 + 5d)\,. \tag{25}$$

In other words, up to a constant additive loss in each interval (t_r, t'_r), a constant fraction of the times are good.

Volume of the set: In order to infer a lower bound on the volume of good times during $[t^-, t^+]$, we subtract from $t^+ - t^-$ the volume of some intervals during which we cannot exclude that a node from W switches to *supp* $\to$ *resync*, increased by the constant term $R_1 + 6\Delta_g + T_1 + 5d$ from (25). The bound then yields that at least a fraction of $(T_2 - 2\vartheta\Delta_g - (\vartheta - 1)T_1 - 2\vartheta d)/(T_2 - (\vartheta - 1)T_1 - \vartheta d)$ of the remaining volume of times is good. Note that we also need to account for the fact that non-faulty nodes may already be in state *supp* $\to$ *resync* at time t^-, which we do by also considering events prior to t^- when non-faulty nodes switch to *supp* $\to$ *resync*. Formally, we define

$$\bar{G} = \bigcup_{\substack{t_r \in [t^- - (R_1 + 6\Delta_g + T_1 + 4d), t^+] \\ \exists i \in W\colon i \text{ switches to } supp \to resync \text{ at } t_r}} [t_r, t_r + R_1 + 6\Delta_g + T_1 + 5d]$$

and strive for a lower bound on the volume of $[t^-, t^+] \setminus \bar{G}$. In order to lower bound the good times in $[t^-, t^+]$, it is thus sufficient to cover all times when a non-faulty node switches to *supp* $\to$ *resync* during $[t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$ by $2N - 1$ intervals of volume at most some $\mathcal{V}$ and then infer a lower bound of $t^+ - t^- - 2N(\mathcal{V} + R_1 + 6\Delta_g + T_1 + 5d)$ on the volume of $[t^-, t^+] \setminus \bar{G}$. The remainder of the proof hence is concerned with deriving such a cover of the times when non-faulty nodes may switch to *supp* $\to$ *resync* during $[t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$.

Observe that nodes in W do not switch to state *init* more than

$$\left\lceil \frac{t^+ - t^- + R_1 + 6\Delta_g + T_1 + 5d}{\text{minimal value of } R_3} \right\rceil \overset{(11)}{\leq} \left\lceil \frac{t^+ - t^- + R_1 + 6\Delta_g + T_1 + 5d}{R_2} \right\rceil \overset{(24)}{\leq} \frac{N}{n - f} \tag{26}$$

times during $[t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$.

Now consider the case that a node in W switches to state *supp* $\to$ *resync* at a time t satisfying that no node in W switched to state *init* during $(t - (8\vartheta + 6)d, t)$. This necessitates that this node observes $n - f$ of its channels in state *supp* during $(t - (2\vartheta + 1)d, t)$, at least $n - 2f \geq f + 1$ of which originate from nodes in W. As no node from W switched to *init* during $(t - (8\vartheta + 6)d, t)$, every non-faulty node that has not observed a node $i \in V \setminus W$ in state *init* at a time from $(t - (8\vartheta + 4)d, t)$ when $(R_2, \textit{supp } i)$ is expired must be in a state whose signal is *none* during $(t - (2\vartheta + 2)d, t)$ due to timeouts. Therefore its outgoing channels are not in state *supp* during $(t - (2\vartheta + 1)d, t)$. By means of contradiction, it thus follows that for each node j of the at least $f + 1$ nodes (which are all from W), there exists a node $i \in V \setminus W$ such that node j resets timeout $(R_2, \textit{supp } i)$ during the time interval $(t - (8\vartheta + 4)d, t)$.

The same reasoning applies to any time $t' \notin (t - (8\vartheta + 6)d, t)$ satisfying that some node in W switches to state *supp* $\to$ *resync* at time t' and no node in W switched to state *init* during $(t' - (8\vartheta + 6)d, t')$. Note that the set of the respective at least $f + 1$ events (corresponding to the at least $f + 1$ nodes from W) where timeouts $(R_2, \textit{supp } i)$ with $i \in V \setminus W$ are reset and the set of the events corresponding to t are disjoint, as

they occur during disjoint time intervals. However, the total number of events where such a timeout can be reset during $[t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$ is upper bounded by

$$|V \setminus W||W| \left\lceil \frac{t^+ - t^- + R_1 + 6\Delta_g + T_1 + 5d}{R_2/\vartheta} \right\rceil \overset{(24)}{<} (f+1)N \ , \tag{27}$$

i.e., the total number of channels from nodes not in W ($|V \setminus W|$ many) to nodes in W multiplied by the number of times an associated timeout can expire at a receiving node in W during $[t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$.

Using that observations, we can now infer that $\bar{G}$ can be covered by less than $2N$ intervals of size $(R_1 + 6\Delta_g + T_1 + 5d) + (8\vartheta + 6)d$ each as follows. By Inequality (26), there are no more than N times $t \in [t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$ when one of the $|W| = n - f$ many non-faulty nodes switches to *init* and thus may cause others to switch to state *supp* $\to$ *resync* at times in $[t, t + (8\vartheta + 6)d]$. Similarly, Inequality (27) shows that the channels from $V \setminus W$ to W may cause at most $N - 1$ such times $t \in [t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$, since any such time requires the existence of at least $f + 1$ events where timeouts $(R_2, \textit{supp}\ i)$, $i \in V \setminus W$, are reset at nodes in W, and the respective events are disjoint. Thus, all times $t_r \in [t^- - (R_1 + 6\Delta_g + T_1 + 5d), t^+]$ when some node $i \in W$ switches to *supp* $\to$ *resync* are covered by at most $2N - 1$ intervals of length $(8\vartheta + 6)d$.

This results in a cover $\bar{G}' \supseteq \bar{G}$ consisting of at most $2N - 1$ intervals that satisfies

$$\mathrm{vol}\left(\bar{G}\right) \leq \mathrm{vol}\left(\bar{G}'\right) < 2N(R_1 + 6\Delta_g + T_1 + (8\vartheta + 11)d) \ .$$

As argued previously, summing over the at most $2N$ intervals that remain in $[t^-, t^+] \setminus \bar{G}'$ and using (25), it follows that the volume of good times during $[t^-, t^+]$ is at least

$$\begin{aligned}
&\frac{T_2 - 2\vartheta\Delta_g - (\vartheta - 1)T_1 - 2\vartheta d}{T_2 - (\vartheta - 1)T_1 - \vartheta d}\left(t^+ - t^- - 2N(R_1 + 6\Delta_g + T_1 + (8\vartheta + 11)d\right) \\
&\overset{(12)}{\geq} \lambda(t^+ - t^- - 2N(R_1 + 6\Delta_g + T_1 + (8\vartheta + 11)d)) \\
&= \lambda\left(t^+ - t^- - 2\left(\frac{\vartheta(t^+ - t^-)}{R_2} + \frac{11}{10}\right)(n - f)(R_1 + 6\Delta_g + T_1 + (8\vartheta + 11)d)\right) \\
&= \lambda\left(1 - \frac{2\vartheta(R_1 + 6\Delta_g + T_1 + (8\vartheta + 11)d)(n - f)}{R_2}\right)(t^+ - t^-) \\
&\quad - \frac{11\lambda(R_1 + 6\Delta_g + T_1 + (8\vartheta + 11)d)(n - f)}{5} \\
&\overset{(10)}{\geq} \lambda^2(t^+ - t^-) - \frac{11(1 - \lambda)R_2}{10\vartheta} \ ,
\end{aligned}$$

as claimed. The lemma follows. □

We are now in the position to prove our second main theorem, which states that a good resynchronization point occurs within $\mathcal{O}(R_2)$ time with overwhelming probability.

THEOREM 4.10. *Denote by* $\hat{E}_3 := \vartheta(R_2 + 3d) + 8(1 - \lambda)R_2 + d$ *the maximal value the distribution* R_3 *can attain plus the at most* d *time until* R_3 *is reset whenever it expires. For any* $k \in \mathbb{N}$ *and any time* t*, with probability at least* $1 - (1/2)^{k(n-f)}$ *there will be a good* W*-resynchronization point during* $[t, t + (k + 1)\hat{E}_3]$.

PROOF OUTLINE. Given that Lemma 4.9 shows that in *any* sufficiently long execution most of the times are good, it suffices to show that the adversary has little chances to prevent that some non-faulty node switches to *init* at one of these many good times.

This is guaranteed by the roughly uniform distribution used for R_3. Since clock drifts will not affect the density of the distribution too much, it ensures roughly uniformly distributed *reference* times when an R_3 timeout will expire, regardless of the execution. This unconditional guarantee makes sure that, no matter what the adversary does, there is an upper bound on the probability mass of possible expiration times the adversary can prevent from being good. In other words, the adversary cannot prevent that there is a constant chance that a non-faulty node succeeds in generating a good synchronization point by switching to *init* when its timeout R_3 expires. Because each expiration time is determined independently, we obtain an overwhelming probability that at least one node succeeds. □

PROOF. Fix some node $i \in W$ and denote by t_0 the infimum of times from $[t, t + (k + 1)\hat{E}_3]$ when node i switches to *init*. We have that $t_0 < t + \hat{E}_3$. By induction, it follows that node i will switch to state *init* at least another k times during $[t, t + (k + 1)\hat{E}_3]$ at the times $t_1 < t_2 < \ldots < t_k$. We claim that each such time t_j, $j \in \{1, .., k\}$ is with probability at least $1/2$ good (and therefore is a good W-resynchronization point).

We prove this by induction on j: As induction hypothesis, suppose for some $j \in \{1, \ldots, k-1\}$ that we showed the statement for $j' \in \{1, \ldots, j-1\}$ and the execution of the system is fixed until time t_{j-1}, i.e., $\mathcal{E}|_{[0,t_{j-1}]}$ is given. Now consider the following set of executions that are extensions of $\mathcal{E}|_{[0,t_{j-1}]}$. Fix any adversarial space that, given the random choices made in $\mathcal{E}|_{[0,t_{j-1}]}$, would indeed result in $\mathcal{E}|_{[0,t_{j-1}]}$ (put simply, consider all possible behaviors of the adversary after time t_{j-1}). Within this space, fix all random choices made until time t_{j-1} to be the same as in $\mathcal{E}|_{[0,t_{j-1}]}$, and fix all random choices non-faulty nodes $i' \neq i$ will make in the future. Note that the execution until now is fully determined until time t_{j-1} and the only free parameter left are the random choices of i after time t_{j-1}.

As the rate of the clock driving node i's R_3 is between 1 and ϑ, and all clocks (clock functions) are already fixed, $t_j > t_{j-1}$ is distributed within a (minimal) interval, call it $[t^-, t^+]$, of size at most

$$t^+ - t^- \leq 8(1-\lambda)R_2 \ .$$

Assume for a moment that $t_j = t^+$. We apply Lemma 4.9 to the corresponding execution[29] $\mathcal{E}'$, showing that the volume of times from $[t^-, t^+]$ that are *not* good in $\mathcal{E}'$ is at most

$$(1-\lambda^2)(t^+ - t^-) + \frac{11(1-\lambda)R_2}{10\vartheta} \ .$$

Now consider the actual random distribution of t_j on $[t^-, t^+]$. Since the clock driving the timeout cannot make progress faster than at rate ϑ and the probability density of R_3 is constantly $1/(8(1-\lambda)R_2)$ (with respect to the clock function C) during $[t^-, t^+]$, we obtain that the probability of t_j not being a good time is upper bounded by

$$\frac{(1-\lambda^2)(t^+ - t^-) + 11(1-\lambda)R_2/(10\vartheta)}{8(1-\lambda)R_2/\vartheta} \leq \vartheta(1-\lambda^2) + \frac{11}{80} \overset{(1)}{<} \vartheta\frac{9}{25\vartheta} + \frac{7}{50} = \frac{1}{2} \ .$$

We define $\mathcal{E}|_{[0,t_j)} := \mathcal{E}'|_{[0,t_j)}$, where in $\mathcal{E}$ at time t_j indeed R_3 expires at node i. This defines a distribution of executions $\mathcal{E}|_{[0,t_j)}$ so that t_j is good with probability at least $1/2$.

We complete our reasoning as follows. The inductive construction above shows that for any adversarial space and regardless of the random choices of other nodes, the

[29] Technically, we still have a distribution of executions, since there are future random choices by i. Note, however, that all these executions are identical until time t^+.

times $t_1, \ldots, t_j$ are each good with independent probability at least $1/2$. Hence, this bound applies simultaneously to all at least $n - f$ nodes in W, and we conclude that we have a good time when a node from W switches to *init* during $[t, t + (k + 1)\hat{E}_3]$ with probability at least $1 - (1/2)^{k(n-f)}$. By the definition of good times, this implies that there is a good W-resynchronization point with at least the stated probability, concluding the proof. □

4.3. Stabilization via Good Resynchronization Points

Having established that eventually a good W-resynchronization point t_g will occur, we turn to proving the convergence of the main routine. Our reasoning proceeds as follows.

— A good resynchronization point t_g results in a clean reset of the *join* and *sleep* $\rightarrow$ *waking* memory flags and starts the (implicit) recovery procedure by letting all non-faulty nodes switch to *passive* (Lemma 4.11).
— Eventually, non-faulty nodes that do not switch to *sleep* must drop out of the basic cycle (Lemma 4.12). If there is such a time, denote by $t_{sleep} > t_g$ the minimal such time.
— The reset of the *sleep* $\rightarrow$ *waking* flags and the definition of a good resynchronization point make sure that no non-faulty node will switch to *active* before T_6 expired after the minimal time $t_s > t_{sleep} > t_g$ when a non-faulty node switches to *sleep* $\rightarrow$ *waking* (Lemma 4.13).
— From Corollary 4.6 and the above, it follows that there is a small time window when all non-faulty nodes (if any) that do not end up in state *recover* switch to *sleep* $\rightarrow$ *waking* (Lemma 4.14). If there are f or less such nodes, all non-faulty nodes end up in *recover*. Otherwise, all non-faulty nodes observe at least $f + 1$ non-faulty nodes in state *sleep* $\rightarrow$ *waking* and switch to *active*, at roughly the same time when the nodes executing the basic cycle switch to *waking*.
— Non-faulty nodes in state *waking* switch to *recover* or *ready* before any non-faulty node leaves state *recover*. Non-faulty nodes in state *recover* switch to *join* before T_3 or T_4 expires at any non-faulty node in *ready*. This follows from the above and Condition 3.3.
— We argue that this implies that when some non-faulty node switches to *accept* before T_5, T_7, or R_1 expire at any non-faulty node, all non-faulty nodes (i.e., those in W) will follow shortly, i.e., a W-quasi-stabilization point is reached. Since all non-faulty nodes switch to *join* or *ready* once the respective timeouts expire, Condition 3.3 ensures that this will eventually happen.

We now implement the above strategy. We start with a few helper statements wrapping up that a good resynchronization point guarantees proper reset of flags and timeouts involved in the stabilization process of the main routine.

LEMMA 4.11. *Suppose t_g is a good W-resynchronization point. Then*

(i) each node $i \in W$ switches to passive *at a time $t_i \in (t_g + 4d, t_g + (4\vartheta + 4)d)$ and observes itself in state* dormant *during $[t_g + 4d, \tau_{i,i}(t_i))$,*
(ii) $\mathrm{Mem}_{i,j,\mathsf{join}}|_{[\tau_{i,i}(t_i), t_{\mathsf{join}}]} \equiv 0$ for all $i, j \in W$, where $t_{\mathsf{join}} \geq t_g + 4d$ is the infimum of all times greater than $t_g - T_1 - 2d$ when a node from W switches to join*,*
(iii) $\mathrm{Mem}_{i,j,\mathsf{sleep}\rightarrow\mathsf{waking}}|_{[\tau_{i,i}(t_i), t_s]} \equiv 0$ for all $i, j \in W$, where $t_s \geq t_g + (2 + 1/\vartheta)T_1 \geq t_g + (1 + 1/\vartheta)T_1 + d$ is the infimum of all times greater or equal to t_g when a node from W switches to sleep $\rightarrow$ waking*,*
(iv) no node from W resets its sleep $\rightarrow$ waking *flags during $[t_g + (1 + 1/\vartheta)T_1 + d, t_g + R_1/\vartheta]$, and*

(v) no node from W resets its join *flags during* $[t_g + (1 + 1/\vartheta)T_1 + d, t_g + R_1/\vartheta]$.

PROOF. All nodes in W switch to state *supp* $\to$ *resync* during $(t_g, t_g + 2d)$ and switch to state *resync* when their timeout of $4\vartheta d$ expires, which does not happen until time $t_g + 4d$. Once this timeout expired, they switch to state *passive* as soon as they observe themselves in state *resync*, i.e., by time $t_g + (4\vartheta + 4)d$. Hence, every node $i \in W$ does not observe itself in state *resync* within $[t_g + 3d, \tau_{i,i}(t_i))$, and therefore is in state *dormant* during $[t_g + 3d, \tau_{i,i}(t_i)]$. This implies that it observes itself in state *dormant* during $[t_g + 4d, \tau_{i,i}(t_i))$, completing the proof of Statement (i).

Moreover, from the definition of a good W-resynchronization point we have that no nodes from W are in state *join* at times in $[t_g - T_1 - 2d, t_{join})$. Statement (ii) follows, as every node from W resets its *join* flags upon switching to state *passive* at time t_i.

Regarding Statement (iii), observe first that no nodes from W are in state *sleep* $\to$ *waking* during $(t_g - d, t_g + (2 + 1/\vartheta)T_1)$ for the following reason: By definition of a good W-resynchronization point no node from W switches to *sleep* during $(t_g - \Delta_g, t_g) \supset (t_g - (2\vartheta + 1)T_1 - 3d, t_g)$. Any node in W that is in states *sleep* or *sleep* $\to$ *waking* at time $t_g - (2\vartheta + 1)T_1 - 3d$ switches to state *waking* before time $t_g - d$ due to timeouts. Finally, any node in W switching to *sleep* at or after time t_g will not switch to state *sleep* $\to$ *waking* before time $t_g + (2 + 1/\vartheta)T_1$. The observation follows.

Since nodes in W reset their *sleep* $\to$ *waking* flags at some time from

$$[t_i, \tau_{i,i}(t_i)] \subset (t_g + 3d, t_g + (4\vartheta + 5)d) \overset{(2)}{\subseteq} (t_g + 3d, t_g + (2 + 1/\vartheta)T_1) \ ,$$

Statement (iii) follows.

Statements (iv) and (v) follow from the fact that all nodes in W switch to state *passive* until time

$$t_g + (4 + 4\vartheta)d \overset{(2)}{\leq} t_g + \left(1 + \frac{1}{\vartheta}\right) T_1 \ ,$$

while timeout $(R_1, \textit{supp} \to \textit{resync})$ must expire first in order to switch to *dormant* and subsequently *passive* again. □

Before we proceed with our third main statement showing eventual stabilization, we make a few more basic observations. Firstly, if non-faulty nodes do not make progress on the basic cycle, they must eventually switch to *recover*, i.e., the timeout conditions ensure detection of deadlocks.

LEMMA 4.12. *For any time t^- and any node in W it holds that it must be in state* recover *or* join *or switch to* sleep *at some time from* $[t^-, t^- + (1 - 1/\vartheta)T_1 + T_2 + T_4 + T_5 + 4d)$.

PROOF. Suppose a node from W is never in state *recover* or *join* during $[t^-, t^- + (1 - 1/\vartheta)T_1 + T_2 + T_4 + T_5 + 4d)$. Thus it may follow transitions along the basic cycle only. Assume w.l.o.g. that the node switched to *sleep* right before time t^-. Thus, it switched to state *accept* beforehand, no later than time $t^- - T_1/\vartheta$. Due to timeouts, it either switched to *recover* at some point in time or switched to *sleep*, *sleep* $\to$ *waking*, *waking*, *ready*, *propose*, *accept*, and finally *sleep* again. At each state, it takes less than d time until a respective timeout is started and it observes itself in the respective state. Hence, the node switches to *recover* or *sleep* before time

$$t^- - \frac{T_1}{\vartheta} + \max\{(2\vartheta + 2)T_1 + 3d, T_2\} + T_4 + T_5 + T_1 + 4d \overset{(3)}{=} t^- + \left(1 - \frac{1}{\vartheta}\right) T_1 + T_2 + T_4 + T_5 + 4d \ ,$$

proving the claim of the lemma. □

Secondly, after a good W-resynchronization point t_g, no node from W will switch to state *join* until either time $t_g + T_7/\vartheta + 4d$ or T_6/ϑ time after the first non-faulty node switched to *sleep* $\rightarrow$ *waking* again after t_g. By proper choice of $T_7 > T_6$ and T_6, this will guarantee that nodes from W do not switch to *join* prematurely during the final steps of the stabilization process.

LEMMA 4.13. *Suppose t_g is a good W-resynchronization point. Denote by t_s the infimum of times greater than t_g when a node in W switches to state* sleep $\rightarrow$ waking *and by t_{join} the infimum of times greater than $t_g - T_1 - 2d$ when a node in W switches to state* join*. Then, starting from time $t_g + 4d$, $tr(\text{recover}, \text{join})$ is not satisfied at any node in W until time*

$$\min\left\{t_g + \frac{T_7}{\vartheta} + 4d, t_s + \frac{T_6}{\vartheta}\right\} < t_{\text{join}} \ .$$

PROOF. By Statements (ii) and (iii) of Lemma 4.11 and Inequality (2), we have that $t_s \geq t_g + 2T_1 + 4d \geq t_g + (4\vartheta + 4)d$ and $t_{join} \geq t_g + 4d$. Consider a node $i \in W$ not observing itself in state *dormant* at some time $t \in [t_g + 4d, t_{join}]$. According to Statements (i) and (ii) of Lemma 4.11, the threshold condition of $f + 1$ nodes memorized in state *join* cannot be satisfied at such a node. By statements (i) and (iii) of the lemma, the threshold condition of $f + 1$ nodes memorized in state *sleep* $\rightarrow$ *waking* cannot be satisfied unless $t > t_s$. Hence, if at time t a node from W satisfies that it observes itself in state *active*, we have that T_6 expired after being reset after time t_s, i.e., $t > t_s + T_6/\vartheta$. Moreover, by Statement (i) of Lemma 4.11, we have that if T_7 is expired at any node in W at time t, it holds that $t > t_g + T_7/\vartheta + 4d$. Altogether, we conclude that $tr(\textit{recover}, \textit{join})$ is not satisfied at any node in W during

$$\left[t_g + 4d, \min\left\{t_g + \frac{T_7}{\vartheta} + 4d, t_s + \frac{T_6}{\vartheta}\right\}\right] \ .$$

In particular, t_{join} must be larger than the upper boundary of this interval, concluding the proof. □

Thirdly, after a good W-resynchronization point, any node in W switches to *recover* or to *sleep* $\rightarrow$ *waking* within a bounded time, and all nodes in W doing the latter will do so in rough synchrony. Using the previous lemmas, we can show that this happens before the transition to *join* is enabled for any node in W.

LEMMA 4.14. *Suppose t_g is a good W-resynchronization point and use the notation of Lemma 4.13. Define $t^+ := t_g + (2 - 1/\vartheta)T_1 + T_2 + T_4 + T_5 + 6d$ and denote by t_{sleep} the infimum of all times greater than $t_g - \Delta_g$ when a node in W switches to* sleep*. Then $t_{\text{sleep}} \geq t_g$ and either*

(i) $t_{\text{sleep}} < t^+$ and at time $t_{\text{sleep}} + 2T_1 + 3d$, any node in W is either in one of the states sleep *or* sleep $\rightarrow$ waking *and observed in* sleep*, or is in* recover *and also observed in* recover*, or*

(ii) all nodes in W are observed in state recover *at time $t^+ + 2T_1 + 3d$.*

PROOF OUTLINE. Lemma 4.11 basically says that around t_g, all non-faulty nodes clear their *join* and *sleep* $\rightarrow$ *waking* flags, and that no non-faulty nodes are in these states close enough to t_g to cause inconsistent flag states (e.g., no non-faulty node memorize a non-faulty node in *sleep* $\rightarrow$ *waking* while other non-faulty nodes do not). From this, Lemma 4.13 essentially derives that in order for a non-faulty node to switch to *join* again, first T_7 must expire at some non-faulty node (which cannot happen before time $t^+ + 2T_1 + 3d$) or a non-faulty node must switch to *sleep* and subsequently a timeout $T_6 > \vartheta(2T_1 + 3d)$ must expire. From Corollary 4.6, we know that while no non-faulty

node is in state *join*, any non-faulty node that switches to *sleep* will result in Statement (i) being true. On the other hand, t^+ is sufficiently large so that no non-faulty node switching to *sleep* until then will result in all non-faulty nodes ending up in state *recover*, which follows from Lemma 4.12 and leads to Statement (ii). □

PROOF. By definition of a good resynchronization point, no non-faulty node switches to *sleep* during $(t_g - \Delta_g, t_g)$, giving that $t_{sleep} \geq t_g$ from which it follows that $t_s \geq t_{sleep}$. We distinguish two cases for t_{sleep}:

Case 1: Assume $t_{sleep} < t^+$. Then Lemma 4.13 yields that

$$t_{join} > \min\left\{t_g + \frac{T_7}{\vartheta} + 4d, t_{sleep} + \frac{T_6}{\vartheta}\right\} \overset{(8)}{\geq} \min\left\{t^+ + 2T_1 + 4d, t_{sleep} + \frac{T_6}{\vartheta}\right\} \overset{(7)}{\geq} t_{sleep} + 2T_1 + 3d\,.$$

Therefore, by definition of a good resynchronization point, no non-faulty node is in state *join* during $(t_g - T_1 - 2d, t_{join}) \supset (t_{sleep} - T_1 - 2d, t_{sleep} + 2T_1 + 3d)$. Recalling that during $(t_g - \Delta_g, t_{sleep})$, no non-faulty node is in state *sleep*, the preconditions of Corollary 4.6 hold, implying Statement (i).

Case 2: Contrary, assume that $t_{sleep} \geq t^+$. By definition of a good resynchronization point no non-faulty node switched to sleep during $(t_g - \Delta_g, t^+) \supset (t_g - T_1 - 2d, t^+)$ and no non-faulty node is in state *join* during $(t_g - T_1 - 2d, t_{join})$. By Lemma 4.13,

$$t_{join} > \min\left\{t_g + \frac{T_7}{\vartheta} + 4d, t_{sleep} + \frac{T_6}{\vartheta}\right\} \overset{(8)}{\geq} \min\left\{t^+ + 2T_1 + 4d, t^+ + \frac{T_6}{\vartheta}\right\} \overset{(7)}{\geq} t^+ + 2T_1 + 3d\,.$$

Hence, Lemma 4.12 states that every non-faulty node must be in state *recover* at some time in $(t_g - T_1 - 2d, t^+ - 2T_1 - 4d)$. If no non-faulty node leaves state *recover* for another d longer, i.e., during $(t_g - T_1 - 2d, t^+ - 2T_1 - 3d)$, all non-faulty nodes will be and be observed in state *recover* by time $t^+ - 2T_1 - 3d$, implying Statement (ii) holds. Otherwise, suppose there is some non-faulty node switching from *recover* to *accept* at some time $t_{accept} \in (t_g - T_1 - 2d, t^+ - 2T_1 - 3d)$. Then there are at least $n - 2f \geq f + 1$ non-faulty nodes in state *accept* at some times from $(t_{accept} - d, t_{accept})$. These nodes switch to *recover* (because no non-faulty node switches to *sleep* before time $t_{sleep} \geq t^+$) during $(t_{accept} - d, t_{accept} + T_1 + d)$ and stay there until at least time $t_{accept} + 2T_1 + 2d \leq t^+$. This entails that the transition from *recover* to *accept* is impossible during $[t_{accept} + T_1 + 2d, t_{join}) \supseteq [t^+ - T_1 - d, t^+)$. Since all non-faulty nodes are in *recover* at least once during $(t_g - T_1 - 2d, t^+ - T_1 - d)$ and no non-faulty node switches to *sleep* (or *join*) during $(t_g - T_1 - 2d, t^+)$, we conclude that all non-faulty nodes are in states *recover* or *accept* during $[t^+ - T_1 - d, t^+)$. However, non-faulty nodes in *accept* at time $t^+ - T_1 - d$ will leave the state before time t^+ because T_1 expires. Altogether, these observations imply that all non-faulty nodes switch to state *recover* by time t^+ and cannot leave it again until time $t_{join} > t^+ + 2T_1 + 3d$, i.e., Statement (ii) holds. □

We have everything in place for proving that a good resynchronization point leads to stabilization within $R_1/\vartheta - 3d$ time.

THEOREM 4.15. *Suppose t_g is a good W-resynchronization point. Then there is a W-quasi-stabilization point during $(t_g, t_g + R_1/\vartheta - 3d]$.*

PROOF OUTLINE. The proof is carried out by carefully distinguishing cases. Figure 15 summarizes the different cases with their most important times and state switches.

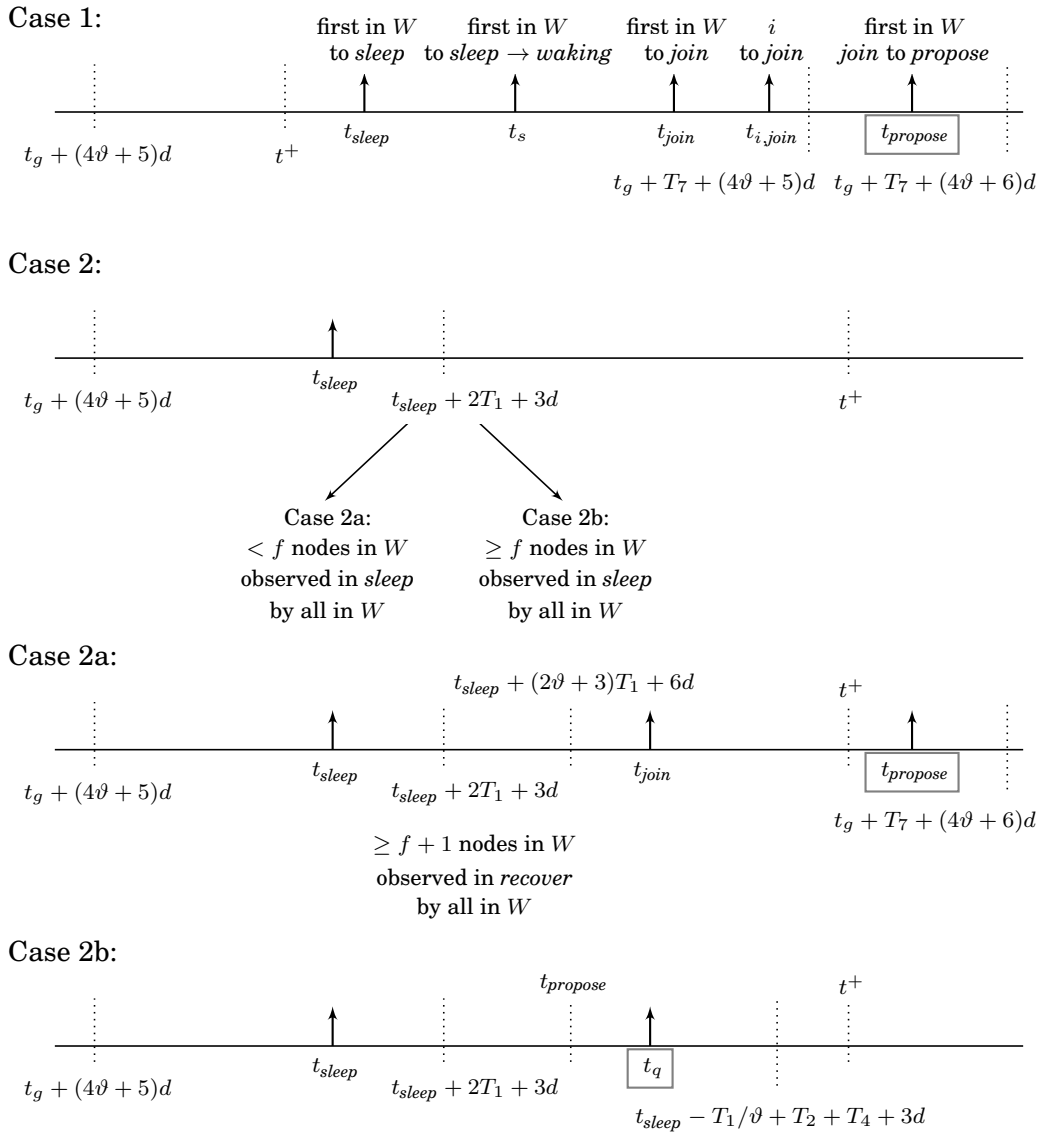

Fig. 15. Visualization of the cases of Theorem 4.15. The times that are shown to be W-quasi-stabilization points are marked with rectangular borders.

By Lemma 4.13, a non-faulty node will eventually switch to *sleep* or all non-faulty nodes end up in *recover*. In the latter case, eventually the T_7 timeouts expire and the system will "reboot" via the *join* state. In case a non-faulty node switches to *sleep* in a timely fashion, the lemma states that shortly after, all non-faulty nodes are in state *recover*, *sleep* or just switched to *sleep* $\rightarrow$ *waking*.

From here, one of the following must be true: (i) at least $f+1$ non-faulty nodes switch to *sleep* $\rightarrow$ *waking* within a small amount of time or (ii) all non-faulty nodes end up in *recover*. This follows from the observation that if at most f non-faulty nodes pass on to *waking*, the remaining at least $n - 2f \geq f + 1$ non-faulty nodes must end up

in *recover*, i.e., non-faulty nodes in *waking* will satisfy $tr(\textit{waking}, \textit{recover})$ and switch to *recover* as well. If (ii) applies, the analysis proceeds as in the previous case. If (i) applies, all non-faulty nodes switch from *passive* to *active* in short order, and therefore start their T_6 timeouts in rough synchrony. This implies that these nodes switch to *join* after non-faulty nodes on the basic cycle switch to *ready*, but before any of them can switch to *propose* due to timeout T_3 (or T_4) expiring. We show that from this configuration, eventually non-faulty nodes will switch to *propose* and *accept* within a window of $3d$ time. The proof is completed by showing that this entire procedure is in all cases complete before R_1 expires at any non-faulty node (which would switch to *dormant* and therefore would be forced to vacate *join*). □

PROOF. For simplicity, assume during this proof that $R_1 = \infty$. Then, by Statement (i) of Lemma 4.11, all nodes in W observe themselves in states *passive* or *active* at times greater or equal to $t_g + (4\vartheta + 5)d$. We will establish the existence of a W-quasi-stabilization point at a time larger than t_g and show that it is upper bounded by $t_g + R_1/\vartheta - 3d$. Hence this assumption can be made w.l.o.g., as the existence of the W-quasi-stabilization point depends on the execution up to time $t_g + R_1/\vartheta$ only, and R_1 cannot expire before this time at any node in W. Moreover, by Statements (i) and (ii) of Lemma 4.11, every node from W satisfies $\mathrm{Mem}_{i,i,\textit{join}} \equiv 0$ on $[t_g + (4\vartheta+5)d, t_{i,\textit{join}})$, where $t_{i,\textit{join}}$ denotes the infimum of all times greater or equal to $t_g + 4d$ when node $i \in W$ switches to *join*. During the time span considered in this proof, every node from W switches at most once to *join*, thus we may w.l.o.g. assume that $\mathrm{Mem}_{i,i,\textit{join}} = 0$ is always satisfied in the following. We use the notation t_s, $t_{\textit{sleep}}$ and t^+ as introduced in Lemmas 4.13 and 4.14. By Statement (iii) of Lemma 4.11 and Inequality (2), we have that $t_s \geq t_g + (1 + 1/\vartheta)T_1 + d \geq t_g + (4\vartheta + 5)d$.

According to Lemma 4.11, all nodes in W switched to state *passive* during $(t_g + 4d, t_g + (4 + 4\vartheta)d)$, implying that at any node in W, T_7 will expire at some time from

$$(t_g + T_7/\vartheta + 4d, t_g + T_7 + (4\vartheta + 5)d) \overset{(8)}{\subset} (t_g + (1 + 1/\vartheta)T_1, t_g + T_7 + (4\vartheta + 5)d) \ .$$

By Lemma 4.13, thus $t_{\textit{join}} > t_g + (1 + 1/\vartheta)T_1$, and by Statement (v) of Lemma 4.11, no node in W resets its *join* flags after $t_g + (1 + 1/\vartheta)T_1$ again (before R_1 expires).

Case 1: Assume $t_{\textit{sleep}} \geq t^+$. Thus, Statement (ii) of Lemma 4.14 applies, i.e., all non-faulty nodes are observed in state *recover* by time $t^+ + 2T_1 + 3d$. Any node from W will switch to state *join* before time $t_g + T_7 + (4\vartheta + 5)d$ because T_7 expires no later than that. Subsequently, it will switch to *propose* as soon as it memorizes all non-faulty nodes in state *join*. Denote by $t_{\textit{propose}} \in (t_g + T_7/\vartheta + 4d, t_g + T_7 + (4\vartheta + 6)d)$ the minimal time when a node from W switches from *join* to *propose*. As observed before, no node in W resets its *join* flags before time $t_{\textit{propose}} + 3d$. As nodes in W reset their *propose* and *accept* flags upon switching to state *join*, some node in W must memorize $n - 2f \geq f + 1$ non-faulty nodes in state *join* at time $t_{\textit{propose}}$. According to Statement (ii) of Lemma 4.11, these nodes must have switched to state *join* at or after time $t_{\textit{join}}$. Hence, all nodes in W will memorize them in state *join* by time $t_{\textit{propose}} + d$ and thus have switched to state *join*. Hence, all nodes in W will switch to state *propose* before time $t_{\textit{propose}} + 2d$ and subsequently to state *accept* before time $t_{\textit{propose}} + 3d$, i.e., $t_{\textit{propose}} \leq t_g + T_7 + (4\vartheta + 6)d$ is a W-quasi-stabilization point.

Case 2: Assume $t_{\textit{sleep}} < t^+$. By Statement (i) of Lemma 4.14, all non-faulty nodes are observed in either *sleep* or *recover* at time $t_{\textit{sleep}} + 2T_1 + 3d$. The nodes observed in state *sleep* will have been observed in state *sleep* $\rightarrow$ *waking* and switched to *waking* by time $t_{\textit{sleep}} + (2\vartheta + 3)T_1 + 4d$.

Case 2a: Suppose less than $f + 1$ nodes in W are observed in state *sleep* at time $t_{\textit{sleep}} + 2T_1 + 3d$ by all non-faulty nodes, i.e., at least $n - 2f \geq f + 1$ non-faulty nodes are

observed in state *recover* by all non-faulty nodes. By Lemma 4.13, we have that

$$\begin{aligned} t_{sleep} + (2\vartheta+3)T_1 + 6d &\overset{(7)}{\leq} \min\left\{t_{sleep} + \frac{T_6}{\vartheta}, t^+ + (2\vartheta+3)\,T_1 + 6d\right\} \\ &\overset{(8)}{\leq} \min\left\{t_s + \frac{T_6}{\vartheta}, t_g + \frac{T_7}{\vartheta} + 4d\right\} \\ &< t_{join}\ . \end{aligned}$$

Hence, any non-faulty node observing itself in state *waking* at some time $t \in (t_{sleep} + 2T_1 + 3d, t_{sleep} + (2\vartheta+3)T_1 + 5d)$ will also observe at least $f+1$ non-faulty nodes in state *recover* and switch to *recover*. As any non-faulty node in *sleep* or *sleep* $\to$ *waking* at time $t_{sleep}+2T_1+3d$ will observe itself in state *waking* no later than time $t_{sleep}+(2\vartheta+3)T_1+5d$, by time $t_{sleep} + (2\vartheta+3)T_1 + 6d < t_{join}$, all non-faulty nodes observe themselves in state *recover*. From here we can argue analogously to the first case, i.e., there exists a W-quasi-stabilization point $t_{propose} \leq t_g + T_7 + (4\vartheta+6)d$.

Case 2b: Suppose at least $f+1$ nodes in W are observed in state *sleep* at time $t_{sleep} + 2T_1 + 3d$ by all non-faulty nodes. These nodes will switch to *waking* and subsequently *ready* until time

$$\max\left\{t_{sleep} + (2\vartheta+3)T_1 + 7d, t_{sleep} - \frac{T_1}{\vartheta} + T_2 + d\right\} \overset{(3)}{=} t_{sleep} - \frac{T_1}{\vartheta} + T_2 + d \tag{28}$$

due to T_2 being expired while observing themselves in *waking* unless they switch from *waking* to *recover*. Note that these nodes reset their *accept* flags upon switching to *waking*. Denote by $t_{propose}$ and t_{accept} the infima of times greater than $t_{sleep} + 2T_1 + 4d$ when a non-faulty node switches to *propose* or *accept*, respectively. Recall that any non-faulty node switching from *recover* to *join* resets its *propose* and *accept* flags, and any non-faulty node switching from *waking* to *ready* resets its *propose* flags. Hence, we have for all $i, j \in W$ that

(i) $\mathrm{Mem}_{i,j,propose}(t) = 0$ at any time $t \in [t_{sleep} + 2T_1 + 3d, t_{propose}]$ at which i observes itself in *ready* or *join*, and
(ii) $\mathrm{Mem}_{i,j,accept}(t) = 0$ at any time $t \in [t_{sleep} + 2T_1 + 3d, t_{accept}]$ at which i observes itself in *ready*, *join*, or *propose*.

By Statements (ii) and (iv) of Lemma 4.11, no node from W resets its *sleep* $\to$ *waking* flags at or after time $t_s \geq t_g + (1+1/\vartheta)T_1 + d$. As $t_s \geq t_{sleep} + 2T_1 + T_1/\vartheta \geq t_{sleep} + 2T_1 + 3d$ and all nodes from W observed in *sleep* at time $t_{sleep} + 2T_1 + 3d$ will be observed in *sleep* $\to$ *waking* by time $t_{sleep} + (2\vartheta+3)T_1 + 4d$, Statement (i) of Lemma 4.11 implies that all nodes in W switch to *active* at some time from $(t_s, t_{sleep} + (2\vartheta+3)T_1 + 4d) \subseteq (t_{sleep}+2T_1+3d, t_{sleep}+(2\vartheta+3)T_1+4d)$. As, by the Statements (i) and (ii) from above, the first node in W switching to state *propose* must do so because of an expiring timeout,

Lemma 4.13 yields that

$$\begin{aligned}
& t_{propose} \\
\geq\ & \min\left\{t_{join}, t_{sleep} - T_1 - d + \frac{T_2 + \min\{T_3, T_4\}}{\vartheta}\right\} \\
\geq\ & \min\left\{t_g + \frac{T_7}{\vartheta} + 4d, t_s + \frac{T_6}{\vartheta}, t_{sleep} - T_1 - d + \frac{T_2 + \min\{T_3, T_4\}}{\vartheta}\right\} \\
\overset{(5)}{\geq}\ & \min\left\{t_{sleep} - t^+ + t_g + \frac{T_7}{\vartheta} + 4d, t_{sleep} + \left(2 + \frac{1}{\vartheta}\right) T_1 + \frac{T_6}{\vartheta}, t_{sleep} - T_1 - d + \frac{T_2 + T_3}{\vartheta}\right\} \\
\overset{(4,8)}{=}\ & t_{sleep} + \left(2 + \frac{1}{\vartheta}\right) T_1 + \frac{T_6}{\vartheta} \,.
\end{aligned}$$

Therefore,

$$t_{propose} \geq t_{sleep} + \left(2 + \frac{1}{\vartheta}\right) T_1 + \frac{T_6}{\vartheta} \overset{(7)}{\geq} t_{sleep} - \frac{T_1}{\vartheta} + T_2 + 2d \,. \tag{29}$$

By Inequality (28), we conclude that at time $t_{sleep} - T_1/\vartheta + T_2 + 2d < t_{propose}$, any node from W observes itself in one of the states *ready*, *recover*, or *join*.

Again, we distinguish two cases.

Case 2b-I: $t_{propose} < t_{sleep} - T_1 - d + (T_2 + T_3)/\vartheta$. As previously used, no non-faulty node can switch from *ready* to *propose* during $(t_{sleep} + 2T_1 + 4d, t_{sleep} - T_1 - d + (T_2 + T_3)/\vartheta))$. Hence, there must be a non-faulty node that switches from *join* to *propose* at time $t_{propose}$. By Statements (i) and (ii) from above, the node must memorize at least $n - 2f \geq f + 1$ nodes from W in state *join* at time $t_{propose}$. By Statement (ii) of Lemma 4.11, these nodes must have switched to *join* at or after time t_{join}. Recall that no non-faulty node resets its *join* flags during $[t_g + (1 + 1/\vartheta)T_1 + d, t_g + R_1/\vartheta)$. Since $_{propose} \geq t_{join} \geq t_g + (1 + 1/\vartheta)T_1 + d$, no non-faulty node resets its *join* flags during $[t_{propose}, t_{propose} + 3d)$. Hence, all non-faulty nodes still in state *recover* have switched to *join* by time $t_{propose} + d$, giving that all non-faulty nodes are in one of the states *ready*, *join*, or *accept* at time $t_{propose} + d$ (since they cannot leave *accept* earlier than $t_{propose} + T_1/\vartheta \geq t_{propose} + 4d$ again).

Case 2b-II: $t_{propose} \geq t_{sleep} - T_1 - d + (T_2 + T_3)/\vartheta$. Recall that all non-faulty nodes switched to *active* by time $t_{sleep} + (2\vartheta + 3)T_1 + 4d$. Hence, any non-faulty node observing itself in state *recover* at time $t_{sleep} + 2T_1 + 3d$ will have switched to *join* because T_6 expired by time

$$t_{sleep} + (2\vartheta + 3)T_1 + T_6 + 5d \overset{(4)}{\leq} t_{sleep} - T_1 - d + \frac{T_2 + T_3}{\vartheta} \leq t_{propose} \,. \tag{30}$$

Hence, also in this case all non-faulty nodes are in one of the states *ready*, *join*, or *accept* at time $t_{propose} + d$.

Continuing Case 2b: Next, we claim that any non-faulty node is in states *propose* or *join* by time $t_{sleep} - T_1/\vartheta + T_2 + T_4 + 3d$. To see this, observe that any non-faulty node following the basic cycle must switch from *ready* to *propose* by this time due to timeouts. On the other hand, according to Inequality (30), all non-faulty nodes in state *recover* switch to *join* by time

$$t_{sleep} - T_1 - d + \frac{T_2 + T_3}{\vartheta} \overset{(3,5)}{<} t_{sleep} - \frac{T_1}{\vartheta} + T_2 + T_4 + 2d \,,$$

showing the claim.

In summary, we showed the following points:

(i) At time $t_{propose} + d$, all non-faulty nodes are observed in states *ready*, *join*, *propose*, or *accept* by all non-faulty nodes.
(ii) All non-faulty nodes switch to *propose* or *join* during $[\min\{t_{join}, t_{propose}\}, t_{sleep} - T_1/\vartheta + T_2 + T_4 + 3d)$.
(iii) No non-faulty node resets its *propose* or *accept* flags at or after time $t_{propose} + d$ unless switching to *accept* first.
(iv) No non-faulty node memorizes non-faulty nodes in state *propose* or *accept* that have not been in that state at or after time $t_{propose}$.

We claim that the infimum t_q of all times from

$$\left[t_{propose}, t_{sleep} - \frac{T_1}{\vartheta} + T_2 + T_4 + 3d\right]$$

when a node from W switches to *accept* is a W-quasi-stabilization point. Note that because

$$t_{sleep} - \frac{T_1}{\vartheta} + T_2 + T_4 + 4d \overset{(6)}{\leq} t_{sleep} + 4d + \frac{T_5 + T_6}{\vartheta} \overset{(29)}{\leq} t_{propose} + \frac{T_5}{\vartheta}$$

no node from W will switch from *propose* to *recover* before time $t_q + 3d$.

Again, we distinguish two cases. First assume that $t_q < t_{sleep} - T_1/\vartheta + T_2 + T_4 + 3d$, i.e., at time t_q indeed a node from W switches to state *accept*. Due to Statement (iv) from the above list and the minimality of t_q, it follows that the respective node memorizes $n - 2f \geq f + 1$ nodes from W in state *propose* that switched to *propose* at or after time t_p. These nodes must be in one of the states *propose* or *accept* during $[t_q, t_q + 3d]$. According to Statement (i) from above, thus all non-faulty nodes still in *ready* will switch to *propose* by time $t_q + d$. By time $t_q + 2d$, all non-faulty nodes in *join* will observe the at least $n - f$ nodes from W in one of the states *join*, *propose*, or *accept*, and hence switch to *propose*. Another d time later, all nodes in W will have switched to *accept*, i.e., t_q is indeed a W-quasi-stabilization point.

On the other hand, if $t_q = t_{sleep} - T_1/\vartheta + T_2 + T_4 + 3d$, Statement (ii) from the above list gives that all nodes from W are in one of the states *join*, *propose*, or *accept* during $[t_q + d, t_q + 3d]$. Therefore, non-faulty nodes will switch from *join* to *propose* and subsequently from *propose* to *accept* until time $t_q + 3d$ as well. Thus, again, t_q is a W-quasi-stabilization point.

It remains to check that in all cases, the obtained W-quasi-synchronization point occurs no later than time $t_g + R_1/\vartheta - 3d$. In Cases 1 and 2a, we have that $t_{propose}$ is a W-quasi-synchronization point with

$$t_{propose} \leq t_g + T_7 + (4\vartheta + 5)d \overset{(9)}{\leq} t_g + \frac{R_1}{\vartheta} - 3d\,.$$

In Case 2b, it holds that t_q is a W-quasi-synchronization point with

$$\begin{aligned} t_q &\leq t_{sleep} - \frac{T_1}{\vartheta} + T_2 + T_4 + 3d \\ &\leq t^+ - \frac{T_1}{\vartheta} + T_2 + T_4 + 3d \\ &= t_g + 2T_1 - \frac{2T_1}{\vartheta} + 2T_2 + 2T_4 + T_5 + 9d \\ &\overset{(9)}{\leq} t_g + \frac{R_1}{\vartheta} - 3d\,. \end{aligned}$$

We conclude that indeed all nodes in W switch to *accept* within a window of less than $3d$ time before, at any node in W, R_1 expires and it leaves state *resync*, concluding the proof. □

Finally, putting together our main theorems and Lemma 3.4, we deduce that the system will stabilize from an arbitrary initial state provided that a subset of $n-f$ nodes in V remains coherent for a sufficiently large period of time.

COROLLARY 4.16. *Let $W \subseteq V$, where $|W| \geq n-f$, and define for any $k \in \mathbb{N}$*

$$T(k) := (k+2)(\vartheta(R_2+3d)+8(1-\lambda)R_2+d)+R_1/\vartheta\ .$$

Then, for any $k \in \mathbb{N}$, the proposed algorithm is a (W, W^2)-stabilizing pulse synchronization protocol with skew $2d$ and accuracy bounds $(T_2+T_3)/\vartheta - 2d$ and T_2+T_4+7d stabilizing within time $T(k)$ with probability at least $1-2^{-k(n-f)}$. It is feasible to pick timeouts such that $T(k) \in \mathcal{O}(kn)$ and $T_2+T_4+7d \in \mathcal{O}(1)$.

PROOF. The satisfiability of Condition 3.3 with $T(k) \in \mathcal{O}(kn)$ and $T_2+T_4+7d \in \mathcal{O}(1)$ follows from Lemma 3.4. Assume that t^+ is sufficiently large for $[t^- + T(k) + 2d, t^+]$ to be non-empty, as otherwise nothing is to show. By definition, W will be coherent during $[t_c^-, t^+]$, with $t_c^- = t^- + \vartheta(R_2+3d)+8(1-\lambda)R_2+d$. According to Theorem 4.10, there will be some good W-resynchronization point $t_g \in [t_c^-, t_c^- + (k+1)(\vartheta(R_2+3d)+8(1-\lambda)R_2+d)]$ with probability at least $1-1/2^{k(n-f)}$. If this is the case, Theorem 4.15 shows that there is a W-stabilization point $t \in [t_g, t^- + T(k)]$. Applying Theorem 4.4 inductively, we derive that the algorithm is a (W, W^2)-stabilizing pulse synchronization protocol with the bounds as stated in the corollary that stabilizes within time $T(k)$ with probability at least $1-2^{-k(n-f)}$. □

4.4. Late Joining and Fast Recovery

An important aspect of combining self-stabilization with Byzantine fault-tolerance is that the system can remain operational when facing a limited number of transient faults. If the affected components stabilize quickly enough, this can prevent future faults from causing system failure. In an environment where transient faults occur according to a random distribution that is not too far from being uniform (i.e., one deals not primarily with bursts), the mean time until failure is therefore determined by the time it takes to recover from transient faults. Thus, it is of significant interest that a node that starts functioning according to the specifications again synchronizes as fast as possible to an existing subset of correct nodes. Moreover, it is of interest that a node that has been shut down temporarily, e.g. for maintenance, can join the operational system again quickly.

In the terms of our analysis, we show that if there is a set W of at least $n-f$ synchronized nodes (i.e., there has been a W-stabilization point), then any further node $i \in V \setminus W$ will synchronize in $\mathcal{O}(1)$ time (i.e., there will be a $(W \cup \{i\})$-stabilization point). Note that this requires coherency of $(W \cup \{i\})$ for $\mathcal{O}(1)$ time only, i.e., there is no hidden dependence on the "slow" R_2 or R_3 timeouts, and that the guarantee is deterministic.

THEOREM 4.17. *Suppose there exists a node i in V and a set $W \subseteq V$, $|W| \geq n-f$, such that there is a W-stabilization point at some time t^- and $W \cup \{i\}$ is coherent during $[t^-, t^- + (1+5/(2\vartheta))R_1]$. Then there is a $(W \cup \{i\})$-stabilization point at some time $t \in [t^-, t^- + (1+5/(2\vartheta))R_1]$.*

PROOF OUTLINE. If node i ever switches to *sleep* in synchrony with the nodes in W, stabilization is achieved. From Lemma 4.5 we deduce that if it switches to *sleep*, it does this in synchrony with the nodes in W. In case i does not switch to *sleep*, Lemma 4.12

states that it will eventually end up in *recover*. However, from *recover* it will follow the nodes in W when they pass through *accept* the next time; the only remaining possibility to delay stabilization further thus is that i switches to *join*. However, from there it would be "picked up" at the latest in the next iteration of the basic cycle by the nodes in W. If this does not happen, it must switch to *propose* fairly soon, and, by the same arguments as above, will either stabilize by switching to *sleep* or end up in *recover* again. This time $\mathrm{Mem}_{i,j,\mathit{join}}$ will be true, as the timeout conditions guarantee that R_1 is not expired yet, i.e., the flag cannot have been reset yet. Therefore, the node cannot switch to *join* again and will stabilize at the next W-stabilization point. □

PROOF. Again, the proof is executed by distinguishing cases. W.l.o.g., we assume for the moment that $W \cup \{i\}$ is coherent during $[t^-, \infty)$ and later show that indeed $t \le t^- + (1 + 5/(2\vartheta))R_1$.

Case 1: Node i does not switch to *supp* $\to$ *resync* during $[t^-, t^- + 2T_2 + 2T_4 + 14d]$. Thus, after R_1 expires at the latest by time $t^- + R_1 + d$, it will observe itself in *dormant* during $[t^- + R_1 + 2d, t^- + 2T_2 + 2T_4 + 14d]$ and therefore not be (or observe itself) in state *join* during $[t^- + R_1 + 3d, t^- + 2T_2 + 2T_4 + 14d]$. By Theorem 4.4 (i), there is a W-stabilization point $t_W \in [t^- + (T_2 + T_3)/\vartheta, t^- + T_2 + T_4 + 5d)$. Subsequently, the nodes in W will switch to *sleep* during $[t_W + T_1/\vartheta, t_W + T_1 + 3d]$. Denote by t_{sleep} the minimum of the respective times. We apply Lemma 4.5 to $W \cup \{i\}$. Thus, at time $t_{\mathit{sleep}} + 2T_1 + 2d$, node i is either in state *recover* and will not leave until the next W-stabilization point (or it switches to *join*), or it is in state *sleep*, *sleep* $\to$ *waking*, or *waking* and resets its timeout T_2 at some time from $[t_W - \Delta_g + T_1/\vartheta - 4d, t_W + 2T_1 + 5d]$.

Case 1a: Node i is in *recover* at time $t_{\mathit{sleep}} + 2T_1 + 2d$. As it cannot switch to *join* until time $t^- + 2T_2 + 2T_4 + 14d$, it will stay in *recover* until the subsequent W-stabilization point $t'_W \in (t_W + (T_2 + T_3)/\vartheta, t_X + T_2 + T_4 + 5d) \subseteq t_W + (T_2 + T_3)/\vartheta, t^- + 2T_2 + 2T_4 + 10d)$ (existing according to Theorem 4.4). By time t'_W, clearly timeout $(\vartheta(2T_1 + 3d), \mathit{recover})$ has expired at node i, as

$$t'_W \ge t_W + \frac{T_2 + T_3}{\vartheta} \overset{(3)}{>} t_{\mathit{sleep}} + 2T_1 + 2d + \vartheta(2T_1 + 3d) + d = t_{\mathit{sleep}} + (\vartheta + 1)(2T_1 + 3d) \; .$$

Because $T_1/\vartheta \ge 4d$, i will observe all nodes from W in *accept* during $[t'_W + 3d, t'_W + 4d]$. Hence it will switch to *accept* by time $t'_W + 3d$, i.e., t'_W is a $(W \cup \{i\})$-quasi-stabilization point.

Case 1b: Node i is in *sleep*, *sleep* $\to$ *waking*, or *waking* at time $t_{\mathit{sleep}} + 2T_1 + 2d$. Denote by t'_W the W-stabilization point subsequent to t_W as in the previous case. As no node from W is in and thus not observed in state *accept* or *recover* during $[t_{\mathit{sleep}} + 2T_1 + 2d, t'_W)$ by Theorem 4.4 (ii) and, as observed before, node i resets its timeout T_2 not earlier than time $t_W - \Delta_g + T_1/\vartheta - 4d$, it will not switch to *recover* before time $\min\{t'_W, t_W - \Delta_g + (T_1 + T_2 + T_3 + T_5)/\vartheta - 4d\}$ unless it switches to *accept* first. By a proof analogous to Lemma 4.5 one can show that i resets its *propose* and *accept* flags before switching to *ready* and d time after the nodes in W have left *accept*. Thus i cannot switch to *accept* before at least f nodes from W switched to *propose* (unless switching to *recover* first). Moreover, by time t'_W, it will already have switched to *ready* since switching there from *sleep* takes at most $T_2 + 3d$ and

$$t'_W \ge t_W + \frac{T_2 + T_3}{\vartheta} \overset{(3,7)}{\ge} t_W + 2T_1 + 5d + T_2 \; .$$

Hence, reasoning analogously to the proof of Theorem 4.4, t'_W is in fact a $W \cup \{i\}$-stabilization point provided that i switches to *accept* instead of *recover* first. This in

turn follows from the bound

$$\begin{aligned} & t_W - \Delta_g + \frac{T_1+T_2+T_3+T_5}{\vartheta} - 4d \\ \overset{(6)}{\geq}\ & t_W - \Delta_g + T_1 + T_2 + T_4 + \frac{T_2+T_3-T_6}{\vartheta} - 4d \\ \overset{(4)}{>}\ & t_W - \Delta_g + (2\vartheta+4)T_1 + T_2 + T_4 + 3d \\ \overset{(2)}{>}\ & t_W + T_2 + T_4 + 7d \\ >\ & t'_W + 2d\,, \end{aligned}$$

where in the last step we used that $t'_W < t_W + T_2 + T_4 + 5d$ according to Theorem 4.4. This shows that T_5 does not expire at i while it is in *propose* before time $t'_W + 2d$. Hence, t'_W is a $W \cup \{i\}$-stabilization point.

Case 2: Node i switches to *supp* $\to$ *resync* at a time $t' \in [t^-, t^- + 2T_2 + 2T_4 + 14d]$. Denote by t_W and t'_W the maximal W-stabilization point smaller than t' and the minimal W-stabilization point larger than $\max\{t', t_W + 2d\}$, which exist by Theorem 4.4. Denote by t_{sleep} the minimal time larger than t_W when a node from W switches to *sleep*. Analogously[30] to Case 1b, t'_W is a $W \cup \{i\}$-stabilization point if i is in state *sleep* at time $t_{\mathit{sleep}} + 2T_1 + 2d$. Hence, assume w.l.o.g. that i is in state *recover* or already switched to *join* by this time. Analogously to Case 1a, t'_W will be a $(W \cup \{i\})$-quasi-stabilization point if it stays in *recover* until time $t'_W + 3d$. Therefore, w.l.o.g., i switches to *join* at some time during $(t', t'_W + 3d)$, implying that it will leave the state no later than time $t'_W + 4d$ and switch to state *accept* by time $t'_W + 5d$.

Now either i continues to execute the basic cycle and thus will, analogously to Case 1b, participate in the minimal W-stabilization point $t''_W > t'_W + 2d$, or it will switch to *recover* again. In the latter case, it cannot switch back to *join* until at least time $t' + R_1/\vartheta$ because it needs to reset its *join* flags first, which happens upon switching to *passive* only. As we have that

$$\begin{aligned} t' + \frac{R_1}{\vartheta} \overset{(9)}{\geq}\ & t' - \frac{2T_1}{\vartheta} + 2T_2 + 2T_4 + T_5 + 7d \\ \overset{(6)}{>}\ & t' - \frac{2T_1}{\vartheta} + 3T_2 + 3T_4 - T_6 + 7d \\ \overset{(4,5)}{>}\ & t' + 2T_2 + 2T_4 + 14d \\ \geq\ & t''_W + 4d\,, \end{aligned}$$

the latter of which follows from applying Theorem 4.4 (ii) two times, i cannot leave state *recover* through *join* again before time $t''_W + 4d$. Therefore, t''_W is a $(W \cup \{i\})$-quasi-stabilization point, analogously to Case 1a.

We have shown that there is some $(W \cup \{i\})$-quasi-stabilization at the latest by time

$$t''_W \leq t' + 2T_2 + 2T_4 + 10d$$

in Case 2, while in Case 1 there is a $(W \cup \{i\})$-quasi-stabilization point no later than time $t^- + 2T_2 + 2T_4 + 14d$. By Theorem 4.4 (ii), this implies a $(W \cup \{i\})$-stabilization point by time

$$t^- + 2T_2 + 2T_4 + 14d + (T_2 + T_4 + 5d) < t^- + \left(1 + \frac{5}{2\vartheta}\right) R_1\,,$$

[30]Note that we can apply Lemma 4.5 even if i switches to *join*, as we can simply replace the set A in the proof of Lemma 4.5 by W, allowing this stronger version of the lemma to hold.

where the bound is obtained analogously to the bound $t' + R_1/\vartheta > t''_W + 4d$ shown above. This concludes the proof, as indeed there is a $(W \cup \{i\})$-stabilization point no later than time $t^- + (1 + 5/(2\vartheta))R_1$. □

5. GENERALIZATIONS

This section provides a few extensions of the core results derived in the previous section. In particular, we show that it is not necessary to map faulty channels to, for example, faulty nodes (thus rendering a non-faulty node effectively faulty in terms of results), that the algorithm can tolerate an even stronger adversary than defined in Section 2 without significant change of stabilization time, and that in many reasonable settings stabilization takes $\mathcal{O}(R_1)$ time only, even if there is no majority of non-faulty nodes that is already synchronized.

5.1. Synchronization Despite Faulty Channels

Theorem 4.15 and our notion of coherency require that all involved nodes are connected by correct channels only. However, it is desirable that non-faulty nodes synchronize even if they are not connected by correct channels. To capture this, the notions of coherency and stability can be generalized as follows.

Definition 5.1 (*Weak Coherency*). We call the set $C \subseteq V$ *weakly coherent* during $[t^-, t^+]$, iff for any node $i \in C$ there is a subset $C' \subseteq C$ that contains i, has size $n - f$, and is coherent during $[t^-, t^+]$.

In particular, if there are in total at most f nodes that are faulty or have faulty outgoing or incoming channels, then the set of all non-faulty nodes is (after some amount of time) weakly coherent.

COROLLARY 5.2. *For each $k \in \mathbb{N}$ let $T'(k) := T(k) - ((\vartheta(R_2 + 3d) + 8(1 - \lambda)R_2 + d))$, where $T(k)$ is defined as in Corollary 4.16. Suppose the subset of nodes $C \subseteq V$ is weakly coherent during the time interval $[t^-, t^+] \supseteq [t^- + T'(k) + T_2 + T_4 + 8d, t^+] \neq \emptyset$. Then, with probability at least $1 - (f + 1)/2^{k(n-f)}$, there is a C-quasi-stabilization point $t \leq t^- + T'(k) + T_2 + T_4 + 5d$.*

PROOF OUTLINE. By definition, each node i is part of a coherent set C_i of at least $n - f$ nodes that, by Corollary 4.16, stabilizes within $T(k)$ time; by the union bound, this happens with probability $1-(f+1)/2^{k(n-f)}$ for *all* these sets. Since the intersection of two such sets C_i, C_j of size at least $n - f$ contains at least $n - 2f \geq f + 1$ nodes, the stabilization points of these sets cannot be far apart. If the nodes in C_i switch to *accept* during some interval $[t, t + 2d)$, all these nodes will have been observed in state *propose* by time $t + 2d$. Hence, as $|C_i \cap C_j| \geq f + 1$, the nodes in C_j that have not done so yet switch to *propose* and subsequently *accept* by time $t + 3d$. □

PROOF. W.l.o.g. we can assume $t^+ = \infty$ during the proof. By the definition of weak coherency, every node in C is in some coherent set $C' \subseteq C$ of size $n - f$. Hence, for any such C' it holds that we can cover all nodes in C by at most $1 + |V \setminus C'| \leq f + 1$ coherent sets $C_1, \ldots, C_{f+1} \subseteq C$. By Corollary 4.16 and the union bound, with probability at least $1 - (f + 1)/2^{k(n-f)}$, for each of these sets there will be at least one stabilization point during $[t^-, t^- + T'(k)]$. Assuming that this is indeed true, denote by $t_{i_0} \in [t^-, t^- + T'(k)]$ the time

$$\max_{i \in \{1,\ldots,f+1\}} \{\max\{t \leq t^- + T'(k) \mid t \text{ is a } C_i\text{-stabilization point}\}\} ,$$

where $i_0 \in \{1, \ldots, f + 1\}$ is an index for which the first maximum is attained and t_{i_0} is the respective maximal time, i.e., t_{i_0} is a C_{i_0}-stabilization point.

Since for all $i \in \{1,\dots,f+1\}$ it holds that $C_i \cap C_{i_0} \neq \emptyset$, and by Theorem 4.4 nodes in C_i only switch to *accept* within a time window of size $2d$, once a C_i-stabilization point has occurred, we conclude that all C_i have stabilization points during $(t_{i_0}-2d, t_{i_0}+2d)$.

Define $t'_{i_0} \in (t_{i_0}+2d, t^- + T'(k) + T_2 + T_4 + 5d]$ to be minimal such that it is another C_{i_0}-stabilization point. Such a time must exist by Theorem 4.4. Now suppose t_a is the minimal time in $(t'_{i_0}-2d, t'_{i_0}+2d)$ when a node from C switches to *accept* and this node is in set C_i for some $i \in \{1,\dots,f+1\}$. As usual, there must be at least $f+1$ non-faulty nodes from C_i in state *propose* at time t_a and by time t_a+d, all nodes from C_i will be observed in either of the states *propose* or *accept*. As $|C_i \cap C_j| \geq f+1$ for any $j \in \{1,\dots,f+1\}$, all nodes in C_j will observe at least $f+1$ nodes in states *propose* or *accept* at times in (t_a, t_a+2d). We have that $t_a \geq t_{i_0} + (T_2+T_3)/\vartheta - 2d$ according to Theorem 4.4. As no nodes switched to state *accept* during $(t_{i_0}+2d, t_a)$ and none of them switch to state *recover* (cf. Theorem 4.4), for any j we can bound

$$t_a + d \geq t_i + d + \frac{T_2+T_3}{\vartheta} > t_j - 3d + \frac{T_2+T_3}{\vartheta} \overset{(4)}{>} t_j + T_2 + 3d \tag{31}$$

that all nodes from C_j are in one of the states *ready*, *propose*, or *accept* at time t_a+d. Hence, they will switch from *ready* to *propose* if they still are in *ready* before time t_a+2d. Less than d time later, all nodes in C_j will memorize all nodes in C_j in state *propose* and therefore switch to *accept* if not done so yet. Since j was arbitrary, it follows that t_a is a C-quasi-stabilization point. □

COROLLARY 5.3. *Suppose $C \subseteq V$ is weakly coherent during $[t^-, t^+]$ and $t \in [t^-, t^+ - (T_2+T_4+8d)]$ is a C-quasi-stabilization point. Then*

(i) all nodes from C switch to accept *exactly once within $[t, t+3d)$;*
(ii) there will be a C-quasi-stabilization point $t' \in [t+(T_2+T_3)/\vartheta, t+T_2+T_4+5d)$ satisfying that no node from C switches to accept *in the time interval $[t+3d, t')$;*
(iii) and each node i, $i \in C$, i's main state machine (Figure 1) is metastability-free during $[t+4d, t'+4d)$.

PROOF. Analogously to the proofs of Theorem 4.4 and Corollary 5.2. □

We point out that one cannot get stronger results by the proposed technique. Even if there are merely $f+1$ failing channels, this can e.g. effectively render a node faulty (as it may never see $n-f$ nodes in states *propose* or *accept*) or exclude the existence of a coherent set of size $n-f$ (if the channels connect $f+1$ disjoint pairs of nodes, there can be no subset of $n-f$ nodes whose induced subgraph contains correct channels only). Stronger resilience to channel faults would necessitate to propagate information over several hops in a fault-tolerant manner, imposing larger bounds on timeouts and weaker synchronization guarantees.

Combination of Corollary 5.2 and Corollary 5.3 finally yields:

COROLLARY 5.4. *Let $C \subseteq V$ be such that, for each $i \in C$, there is a set $C_i \subseteq C$ with $|C_i| = n-f$, and let $E = \bigcup_{i \in C} C_i^2$. Then, for any $k \in \mathbb{N}$, the proposed algorithm is a (C,E)-stabilizing pulse synchronization protocol with skew $3d$ and accuracy bounds $(T_2+T_3)/\vartheta - 3d$ and T_2+T_4+8d stabilizing within time $T(k)+T_2+T_4+5d$ with probability at least $1-(f+1)/2^{k(n-f)}$.*

PROOF. Analogously to the proof of Corollary 4.16. □

5.2. Stronger Adversary

So far, our analysis considered a fixed set C of coherent (or weakly coherent) nodes. But what happens if whether a node becomes faulty or not is not determined upfront,

but depends on the execution? Phrased differently, does the algorithm still stabilize quickly with a large probability if an adversary may "corrupt" up to f nodes, but may decide on its choices as time progresses, fully aware of what happened so far? Since we operate in a system where all operations take positive time, it might even be the case that a node might fail just when it is about to perform a certain state transition, and would not have done so if the execution had proceeded differently. Due to the way we use randomization, this however makes little difference for the stabilization properties of the algorithm.

COROLLARY 5.5. *Suppose at every time t, an adversary has full knowledge of the state of the system up to and including time t, and it might decide on in total up to f nodes (or all channels originating from a node) becoming faulty at arbitrary times. If it picks a node at time t, it fully controls its actions after and including time t. Furthermore, it controls delays and clock drifts of non-faulty components within the system specifications, and it initializes the system in an arbitrary state at time 0. For any $k \in \mathbb{N}$, define*

$$t_k := (k+3)(\vartheta(R_2+3d)+8(1-\lambda)R_2+d)+R_1/\vartheta+T_2+T_4+5d\,.$$

Then the set of all nodes that remain non-faulty until time t_k reaches a quasi-stabilization point during $[\vartheta(R_2+3d)+8(1-\lambda)R_2+d, t_k]$ with probability at least

$$1-(f+1)e^{-k(n-f)/2}\,.$$

Moreover, at any time $t \geq \vartheta(R_2+3d)+8(1-\lambda)R_2+d$, the set of nodes that are non-faulty at time t is coherent.

PROOF. The last statement of the corollary holds by definition.

We need to show that Theorem 4.10 holds for the modified time interval $[\vartheta(R_2+3d)+8(1-\lambda)R_2+d, (k+3)(\vartheta(R_2+3d)+8(1-\lambda)R_2+d)]$ with the modified probability of at least $1-e^{-k(n-f)/2}$. If this is the case, we can proceed as in Corollaries 5.2 and 5.3.

We start to track the execution from time $\vartheta(R_2+3d)+8(1-\lambda)R_2+d$. Whenever a non-faulty node switches to state *init* at a good time, the adversary must corrupt it in order to prevent subsequent deterministic stabilization. In the proof of Theorem 4.10, we showed that for any non-faulty node, there are at least $k+1$ different times during $[\vartheta(R_2+3d)+8(1-\lambda)R_2+d, (k+3)(\vartheta(R_2+3d)+8(1-\lambda)R_2+d)]$ when it switches to *init* that have an independently by $1/2$ lower bounded probability to be good. Since Lemma 4.9 holds for *any* execution where we have at most f faulty nodes, the adversary corrupting some node at time t affects the current and future trials of that node only, while the statement still holds true for the non-corrupted nodes. Thus, the probability that the adversary may prevent the system from stabilizing until time t_k is upper bounded by the probability that $(k+1)(n-f)$ independent and unbiased coin flips show f or less times tail. Chernoff's bound states for the random variable X counting the number of tails in this random experiment that for any $\delta \in (0,1)$,

$$P[X < (1-\delta)\mathbb{E}[X]] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mathbb{E}[X]} < e^{-\delta\mathbb{E}[X]}\,.$$

Inserting $\delta = k/(k+1)$ and $\mathbb{E}[X] = (k+1)(n-f)/2$, we see that the probability that

$$P[X \leq f] \leq P[X < (n-f)/2] < e^{-k(n-f)/2}\,,$$

as claimed. □

5.3. Constant-Time Stabilization

Up to now, we considered worst-case scenarios only. In practice, it is likely that faulty nodes show not entirely arbitrary / malicious behavior. In particular, they might still be partially following the protocol, not exhibit a level of coordination that could only be achieved by a powerful central instance, or not be fully aware of non-faulty nodes' states. Moreover, it is unlikely that at the time when a majority of the nodes becomes non-faulty, all their timeouts R_2 and R_3 have been reset recently. In such settings, stabilization will be much easier and therefore be achieved in constant time with a large probability. It is difficult, however, to name simple conditions that cover most reasonable cases. Generally speaking, once (randomized) timeouts of duration R_2 or R_3 are not "messed up" at non-faulty nodes anymore, faulty channels and nodes need to collaborate in an organized manner in order to prevent stabilization for a large time period. We give a few examples in the following corollary.

COROLLARY 5.6. *Suppose $W \subseteq V$, where $|W| \geq n - f$, satisfies that for each $i \in W$, all (randomized) timeouts of duration R_2 or R_3 are correct during $[t^-, t^+]$, and the node is non-faulty during $[t^- + \vartheta(3(R_2+3d)+2(8(1-\lambda)R_2+d)), t^+]$. Moreover, channels between nodes in W are correct during $[t^- + \vartheta(3(R_2+3d)+2(8(1-\lambda)R_2+d)), t^+]$ and did not insert* init *signals that have not been sent during $[t^-, t^- + \vartheta(3(R_2+3d)+2(8(1-\lambda)R_2+d))]$ or delay them by more than R_1 time. Define $\tilde{t}^- := t^- + \vartheta(3(R_2+3d)+2(8(1-\lambda)R_2+d)) + R_1 + d$. Moreover, assume that one of the following statements holds during $[\tilde{t}^-, t^+]$.*

(i) Nodes in $V \setminus W$ switch to init *at times that are independently distributed with probability density at most $\mathcal{O}(1/(R_1 n))$, and channels from $V \setminus W$ to W do not generate* init *signals on their own (or delay* init *signals from before $\tilde{t}^-$ more than R_1 time).*

(ii) Channels from $V \setminus W$ to W switch to init *at times that are independently distributed with probability density at most $\mathcal{O}(1/(R_1 n^2))$.*

(iii) Channels from $V \setminus W$ to W switch to init *obliviously of the history of signals originating at nodes in W and do not know the time $\tilde{t}^-$.*

If $t^+ \in \tilde{t}^- + \Omega(kR_1)$, $k \in \mathbb{N}$, then there is a W-stabilization point during $[\tilde{t}^-, \tilde{t}^- + \mathcal{O}(kR_1)]$ with probability at least $1 - 2^{-\Omega(k)}$.

PROOF. In Theorems 4.4 and 4.15, we showed that stabilization is deterministic once a good resynchronization point occurs. The notion of coherency essentially states that at non-faulty nodes, each timeout expired at least once and has not been reset again because of incorrect observations on other non-faulty nodes' states until the set is considered coherent (cf. Lemma 4.3). Subsequently, the respective nodes are non-faulty and the channels connecting them correct. This is true by the prerequisites of the corollary, which essentially state respective conditions on timeouts R_2 and R_3 explicitly, while rephrasing the conditions for coherency for the remaining timeouts (note that R_1 is the largest timeout except for R_2 and R_3).

Moreover, the time span during which R_2 and R_3 behave and are observed regularly is large enough for R_3 to expire twice and additional R_2+3d time to pass. This accounts for the fact that in the proof of Theorem 4.10, we essentially first wait until R_3 expires once (so the adversary has no useful information on the timeout at the respective node anymore) and then consider the subsequent time(s) when it expire(s). The proof then exploits that non-faulty nodes timeout R_3 will expire at roughly independently uniformly distributed points in time. Therefore, unless faulty nodes or channels interfere, the statement of the corollary holds.

Hence, we need to show that for any of the three conditions, there is not too much meddling from outside W. For Conditions (i) or (ii), we see that the probability that there are no *init* signals on channels from $V \setminus W$ to W at all for any time span of

length $\mathcal{O}(R_1)$ is at least constant, regardless of the time interval considered. Regarding Condition (iii), recall that Theorem 4.10 essentially shows that whatever the strategy of the adversary, the expected number of good W-resynchronization points during a time interval (where W is coherent) is linear in the size of the interval divided by R_1 if the interval is sufficiently large. Since the adversary is oblivious of the current time in relation to $\hat{t}^+$ and the state of W, the statement that for any strategy of the adversary the amortized number of good stabilization points per R_1 time is constant yields the claim of the lemma. □

We remark that this observation is particularly interesting as the core routine of the algorithm is independent of the resynchronization routine after stabilization. If at some time W becomes subject to a large number of faults resulting in loss of synchronization, however the resynchronization routine still works properly, it is very likely that W will recover within $\mathcal{O}(1)$ time (provided $R_1 \in \mathcal{O}(1)$). On the other hand, if the resynchronization routine fails in the sense that a majority of the nodes suffers from faulty timeouts R_2 or R_3, or communication is faulty between too many nodes, this will not affect the core routine unless too many components related to it fail as well.

6. THE FATAL$^+$ PROTOCOL

As discussed in the introduction, once pulse synchronization is established, the pulses may serve as round delimiters for the execution of synchronous rounds. Applying the algorithm by Hoch et al. [2006], these rounds can be consistently labeled by integers that increase by one in every round, i.e., fully synchronous executions become feasible.

However, utilizing the pulse synchronization routine as-is will result in poor performance. Firstly, the algorithm due to Hoch et al. [2006] involves the use of shared coins, which is highly expensive in terms of computation and communication. Secondly, despite the fact that the time between pulses is $\Theta(d)$ (if the timeouts are chosen respectively) and thus the time for a simulated round would be asymptotically optimal, system speed would be several orders of magnitude from the lower bound [Lundelius and Lynch 1984], due to the large implied constants (cf. Lemma 3.4). Moreover, our system model assumes that delays may vary arbitrarily between 0 and d. While this is very flexible and helpful when it comes to the—in terms of hardware implementation—fairly complex implementation of communicating the core state machine's states, the pure *wire* delays of the communication channels between different nodes are much smaller and vary within a smaller range.

This section contains an extension of FATAL, termed FATAL$^+$, which overcomes these limitations. In a nutshell, it consists of adding a fast non-self-stabilizing, Byzantine-tolerant algorithm termed *quick cycle* to FATAL, which generates exactly $M > 1$ fast clock ticks between any two pulses at a correct node after stabilization.

The Quick Cycle Algorithm

Consider a system of n nodes, each of which runs the FATAL pulse synchronization protocol. Additionally, each node is equipped with an instance of the *quick cycle* state machine depicted in Figure 16 and an instance of the *cycle counter algorithm* depicted in Figure 17, generating the NEXT signal. Note, that the latter is not stated in terms of a state machines, in order not to distract from its essentials. Although one can easily come up with an equivalent state machine description within our system model, a real-world hardware implementation would rather implement it by a simple adder circuit than a state machine. The interface between the quick cycle algorithm and the underlying FATAL pulse synchronization protocol is by means of two signals only, one for each direction of the communication: (i) The quick cycle state machine and the cycle counter algorithm generate the NEXT signal by which they (weakly) influence the

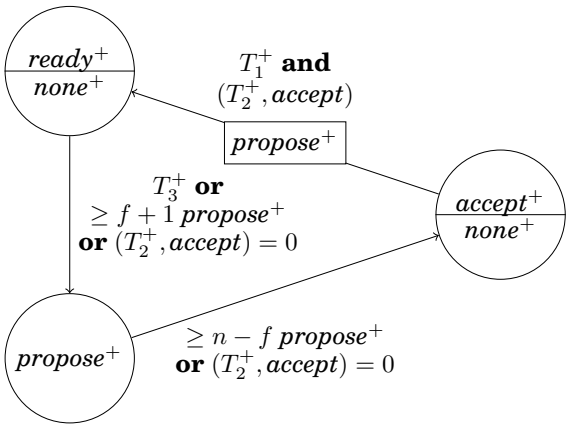

Fig. 16. The quick cycle of the FATAL$^+$ protocol.

Fig. 17: Cycle counter algorithm, also generating the NEXT signal.

Upon (T_2^+, accept) *expires* **do**
  count $\leftarrow 0$
  generate NEXT pulse
end
Upon *switch to* accept$^+$ **do**
  count $\leftarrow$ count $+ 1 \mod M$
  if *count* $= 0$ **then**
    generate NEXT pulse
  end
end

time between two successive pulses generated by FATAL, and (ii) the quick cycle state machine observes the state of the (T_2^+, accept) signal, which signals the expiration of an additional timer added to the FATAL protocol. The timer is coupled to the state *accept* of FATAL, in which the pulse synchronization algorithm generates a new pulse. The signal's purpose is to enforce a consistent reset of the quick cycle state machine once FATAL has stabilized.

Essentially, the quick cycle state machine is a copy of the outer cycle of Figure 3 that is stripped down to the minimum. However, an additional mechanism is introduced in order to ensure stabilization, namely, some coupling to the *accept* state of the main algorithm: Whenever a pulse is generated by FATAL, we require that all non-faulty nodes switch to the *accept*$^+$ state unless they already occupy that state. This is easily achieved by incorporating the state of the expiration signal of the additional FATAL timer (T_2^+, accept) in the guards of Figure 16.[31] Since pulses are synchronized up to the skew Σ of the pulse synchronization routine, it follows that all non-faulty nodes switch to *accept*$^+$ within a time window of $\Sigma + 2d$: one d for (T_2^+, accept) to be reset and an additional d for a node in *ready*$^+$ switching to *propose*$^+$ by time $\Sigma + d$ to observe itself in *propose*$^+$. Subsequently, all non-faulty nodes will switch to state *ready*$^+$ before the first one switches to *propose*$^+$ provided that T_3^+ is sufficiently large, and the condition that $f+1$ *propose*$^+$ signals trigger switching to *propose*$^+$ guarantees that all non-faulty nodes switch to *accept*$^+$ in a tightly synchronized fashion.

The cycle counter algorithm depicted in Figure 17 maintains an integer cycle counter that nodes increase by one whenever they switch to *accept*$^+$. The counter is reset to zero whenever (T_2^+, accept) expires, i.e., shortly after a pulse generated by the underlying pulse synchronization algorithm. The algorithm makes sure that, once the compound algorithm stabilized, these resets never happen when the counter holds a non-zero value. The counter operates mod $M \in \mathbb{N}$, where M is large enough so that at least roughly $T_2 + T_3$ and at most roughly $(T_2 + T_4)/\vartheta$ time passed since the most recent pulse when it reaches $M \equiv 0$ again. Whenever the counter is set to 0, a non-faulty node $i \in V$ will set its NEXTi signal to 1 and switch it back to 0 at once (thus raising the respective NEXTi memory flag of the main algorithm). Thus, by actively triggering the next pulse, we ensure that a pulse does not occur at an inconvenient point in

[31] Recall that $(T_2^+, \mathit{accept}) = 0$ from the time the timer is reset (i.e., at most d time after the node switches to state *accept* of the main state machine) until it expires, from where on $(T_2^+, \mathit{accept}) = 1$ until it is reset again.

time: When the system has stabilized, exactly M switches to *accept*$^+$ of the quick cycle algorithm occur between any two consecutive pulses at a non-faulty node. As these switches occur also synchronously at different non-faulty nodes, it is apparent that the quick cycle state machine in fact implements a bounded-size synchronized clock.

To derive accurate bounds on the skew of the protocol, we need to state the involved delays more carefully.

Definition 6.1 (*Refined Delay Bounds*). The state of the quick cycle algorithm is communicated via separate channels $S^+_{i,j}$, with $i,j \in V$, whose delays vary within $d^+_{\min}$ and $d^+_{\max}$ in order to be considered correct during $[t^-,t^+]$. State transitions of the quick cycle state machine, resets of its timeouts, and clearance of its memory flags take at most $d^+_{\max}$ time.

Setting $\Sigma^+ := 2d^+_{\max} - d^+_{\min}$, we assume that the following timing constraints hold:

$$T_1^+ \geq \vartheta(T_2^+ + \Sigma^+ + 3d + d^+_{\max}) \tag{32}$$

$$T_2^+ \geq \vartheta(3d + 3d^+_{\max}) \tag{33}$$

$$T_3^+ \geq \vartheta(T_1^+ + d^+_{\max}) \tag{34}$$

$$M \in \left[\frac{\vartheta(T_2 + T_3 + 3d) + T_1^+ - T_2^+}{T_1^+ + T_3^+}, \frac{T_2 + T_4 - 3\vartheta d}{\vartheta(T_1^+ + T_3^+ + \Sigma^+ + 4d^+_{\max})} \right] . \tag{35}$$

It follows from Lemma 3.4 that it is always possible to pick appropriate values for the timeouts and M. Note, however, that choosing $M \in \omega(1)$ requires that $T_2 + T_3 \in \omega(1)$, resulting in a superlinear stabilization time. More precisely, the stabilization time of FATAL$^+$ is, given M and minimizing the timeouts under this constraint, in $\Theta(Mn)$.

We now prove the correctness of the FATAL$^+$ protocol.

THEOREM 6.2. *Let $W \subseteq V$, where $|W| \geq n - f$, and define $T(k)$, for $k \in \mathbb{N}$, as in Corollary 4.16. Then, for any $k \in \mathbb{N}$, the FATAL$^+$ protocol is a (W, W^2)-stabilizing pulse synchronization protocol (where* accept$^+$ *is the "pulse" state) with skew Σ^+ and accuracy bounds $(T_1^+ + T_3^+)/\vartheta - \Sigma^+$ and $T_1^+ + T_3^+ + 2\Sigma^+ + 3d^+_{\max}$. It stabilizes within time $T(k) + T_1^+ + T_3^+ + \Sigma^+ + 3d + 5d^+_{\max}$ with probability at least $1 - 2^{-k(n-f)}$. Moreover, the cycle counters increase by exactly one mod M at each pulse, within a time window of Σ^+, and both the quick cycle state machine and the cycle counters are metastability-free once the protocol stabilized and remains fault-free in W.*

PROOF OUTLINE. Once the underlying FATAL protocol stabilized (within $T(k)$ time), the $(T_2^+, \textit{accept})$ signals will be well-synchronized at all non-faulty nodes. Hence they will force a consistent (re)start of the system, no matter what the initial configuration was. Once these timeouts expire after a short amount of time, the non-faulty nodes will just execute the quick cycle, resynchronizing on each transition to *propose*$^+$, without interference by any non-expired $(T_2^+, \textit{accept})$ signal. M is chosen such that this clock value will be reached before T_4 expires at non-faulty nodes in the main state machine. Hence, by the time when their main state machines switch to state *propose*, the non-faulty nodes will have (almost) completed their last iteration of the quick cycle before the wrap-around of the clock (i.e., their counters equal $M-1$ or 0 already); at the same time, T_3 will be already expired at these nodes, implying that they all switch to *propose* and subsequently *accept*. Thus, the "reset" of the quick cycle simply coincides with the wrap-around of the clock to 0, i.e., the operation of the quick cycle is not disturbed.

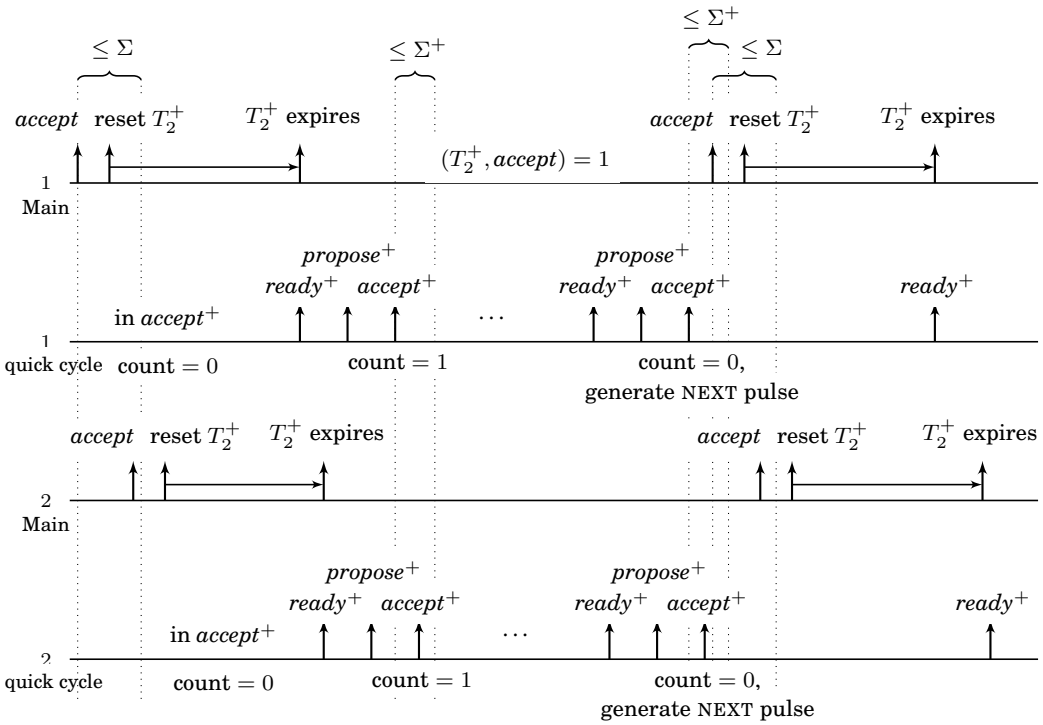

Fig. 18. Part of an execution showing the interaction of the main state machine of FATAL and the quick cycle at nodes 1 and 2.

Figure 18 shows part of an execution of FATAL$^+$ that has already stabilized. Note, the potentially smaller skew Σ^+ of FATAL$^+$ in comparison to the skew Σ of the underlying FATAL protocol. □

PROOF. Assume that nodes in $W \subseteq V$, where $|W| \geq n - f$, are non-faulty and channels between them are correct during $[t^-, t^+]$, where $t^+ \geq t^- + T(k) + T_1^+ + T_3^+ + \Sigma^+ + 3d + 3d_{\max}^+$. According to Corollary 4.16, with probability at least $1 - 2^{-k(n-f)}$, there exists a time $t_0 \in [t^-, t^- + T(k))]$ such that all nodes in W switch to *accept* within $[t_0, t_0 + 2d)$, and they will continue to switch to *accept* regularly in a synchronized fashion until at least t^+. For the remainder of the proof, we assume that such a time t_0 is given; from here we reason deterministically.

The skew bound is shown by induction on the k-th consecutive quick cycle pulse, where $k \in \mathbb{N}$, generated after the stabilization time t_0 of the FATAL algorithm. Note that the time for which we are going to establish that the compound algorithm stabilizes is $t_1 > t_0$; here we denote for $k \geq 1$ by t_k the time when the first node from W switches to *accept*$^+$ for the k^{th} time after $t_0 + 3d$, i.e., the beginning of the k^{th} pulse of FATAL$^+$ that we prove correct. W.l.o.g. we assume that $t^+ = \infty$; otherwise, all statements will be satisfied until t^+ only (which is sufficient).

To prove the theorem, we are going to show by induction on $k \in \mathbb{N}$ that

(i) $t_1 \in [t_0 + (T_2^+ + T_3^+)/\vartheta, t_0 + T_1^+ + T_3^+ + \Sigma^+ + 3d + 5d_{\max}^+]$,
(ii) if $k \geq 2$, $t_k \leq t_{k-1} + T_1^+ + T_3^+ + \Sigma^+ + 3d_{\max}^+$,
(iii) if $k \geq 2$, $t_k \geq t_{k-1} + (T_1^+ + T_3^+)/\vartheta$,

(iv) $\forall i \in W : i$ switches to *accept*$^+$ for the k^{th} time after $t_0 + 3d$ during $[t_k, t_k + \Sigma^+)$,
(v) if $k \geq M$, for $l := \lfloor k/M \rfloor$, $\forall i \in W : i$ switches to *accept* for the l^{th} time after $t_0 + 3d$ during $[t_{Ml}, t_{Ml} + \Sigma^+ + 2d)$,
(vi) $\forall i \in W$: node i's cycle counter switches from $k - 1 \bmod M$ to $k \bmod M$ at some time from $[t_k, t_k + \Sigma^+)$, and
(vii) if $k \geq 2$, $\forall i \in W$: node i's cycle counter changes its state exactly once during $[t_{k-1}, t_k)$.

In particular, the protocol is a pulse synchronization protocol with the claimed bounds on skew, accuracy, and stabilization time. Proving these properties will also reveal that quick cycle is metastability-free after time t_1.

To anchor the induction at $k = 1$, we need to establish Statement (i) as well as Statements (iv) and (vi) for $k = 1$; the remaining statements are empty for $k = 1$.

Recall that any node $i \in W$ switches to *accept* during $[t_0, t_0 + 2d)$. Hence, during

$$\left[t_0 + 3d, t_0 + \frac{T_2^+}{\vartheta}\right) \overset{(33)}{\subseteq} [t_0 + 3d, t_0 + 3d + 3d_{\max}^+) \ ,$$

at no node in W, $(T_2^+, \textit{accept})$ is expired, implying that all nodes in W are in state *accept*$^+$ during $[t_0 + 3d + 2d_{\max}^+, t_0 + 3d + 3d_{\max}^+)$. Note that each node in W will reset its cycle counter to 0 when $(T_2^+, \textit{accept})$ expires, i.e., after having completed its transition to *accept*$^+$.

The above bound shows that at the minimal time after $t_0 + 3d$ when a node in W switches to *ready*$^+$, it is guaranteed that no node in W is observed in *propose*$^+$ until the minimal time $t_p \geq t_0 + 3d$ when a node in W switches to *propose*$^+$. Moreover, at any node in W switching to state *ready*$^+$ timeout $(T_2^+, \textit{accept})$ must be expired, implying that the node may not switch from *ready*$^+$ to *propose*$^+$ due to this signal until it switches to *accept* again. Recall that nodes from W set their NEXT signals to 1 only briefly when their cycle counters are set to 0. Hence, for each such node in W, this signal is observed in state 0 from the time when $(T_2^+, \textit{accept})$ expires until (a) at least time t_M or (b) the time the node is forced by a switch to *accept* to set its counter to 0, whatever is earlier. Examining the main state machine, it thus can be easily verified that no node in W may switch from *ready*$^+$ to *propose*$^+$ because $(T_2^+, \textit{accept}) = 0$ before (a) time t_M or (b) time

$$t_0 + \frac{T_2 + T_4}{\vartheta} \overset{(35)}{>} t_0 + M(T_1^+ + T_3^+ + \Sigma^+ + 4d_{\max}^+) + 3d \geq t_0 + M(T_1^+ + T_3^+ + 5d_{\max}^+) + 3d \quad (36)$$

is reached. We obtain:

(P1) No node in W observes $(T_2^+, \textit{accept})(t) = 0$ at some time $t \in [t_0 + 3d, \min\{t_M, t_0 + M(T_1^+ + T_3^+ + 5d_{\max}^+) + 3d\}]$ when it is not in state *accept*$^+$.

Considering that any node $i \in W$ will switch to *ready*$^+$ once both T_1^+ and T_2^+ expired and subsequently to *propose*$^+$ at the latest when T_3^+ expires (provided that it does not switch back to *accept*$^+$ first), it follows that by time

$$\begin{aligned} & t_0 + 3d + \max\{T_1^+ + 3d_{\max}^+, T_2^+\} + T_3^+ + 2d_{\max}^+ \\ \overset{(32)}{=}\ & t_0 + T_1^+ + T_3^+ + 5d_{\max}^+ + 3d & (37) \\ \overset{(36)}{<}\ & t_0 + \frac{T_2 + T_4}{\vartheta} \ , & (38) \end{aligned}$$

each node in W must have been observed in *propose*$^+$ at least once. On the other hand, as we established that nodes do not observe nodes in W in state *propose*$^+$ when switching to *ready*$^+$ at or after time $t + 3d$ before the first node in W switches to *propose*$^+$, it

follows that until time

$$t_0 + \frac{T_2^+ + T_3^+}{\vartheta} \overset{(34),(33)}{\geq} t_0 + 3d + T_1^+ + 2d_{\max}^+ \,, \tag{39}$$

nodes in W will have at most $|V \setminus W| \leq f$ of their *propose*$^+$ flags in state 1, and their timeout T_3^+ did not expire yet. Thus, by (P1), the first node in W that switches to *propose*$^+$ after $t_0 + 3d$ must do so at time $t_p \geq t_0 + 3d + T_1^+ + 2d_{\max}^+$.

Recall that t_1 is the minimal time larger than $t_0 + 3d$ when a node in W switches to state *accept*$^+$. By (37) and since $|W| \geq n - f$, we have that each node in W observes at least $n - f$ nodes in *propose*$^+$ by time $t_0 + T_1^+ + T_3^+ + 3d + 5d_{\max}^+$, and thus

$$t_1 \leq t_0 + T_1^+ + T_3^+ + 3d + 5d_{\max}^+ \,. \tag{40}$$

Moreover, we can trivially bound

$$t_1 \geq t_p \geq t_0 + (T_2^+ + T_3^+)/\vartheta \,. \tag{41}$$

From (40) and (41) it follows that t_1 satisfies Statement (i) of the claim.

Since at time t_1 there is a node $i \in W$ switching from *propose*$^+$ to *accept*$^+$, (P1) implies that it must memorize at least $n - 2f \geq f + 1$ nodes in W in state *propose*$^+$, which must have switched to this state during $[t_p, t_1 - d_{\min}^+]$. By the above considerations regarding the reset of the *propose*$^+$ flags, this yields that all nodes in W will memorize at least $f + 1$ nodes in state *propose*$^+$ by time $t_1 + d_{\max}^+ - d_{\min}^+$ and thus switch to *propose*$^+$ (if they have not done so yet). It follows that by time $t_1 + 2d_{\max}^+ - d_{\min}^+ = t_1 + \Sigma^+$, all nodes in W memorize at least $|W| \geq n - f$ nodes in *propose*$^+$ and therefore switched to *accept*$^+$. Hence, we successfully established Statement (iv) of the claim for $k = 1$. Statement (vi) follows for $k = 1$, as the cycle counters have been reset to zero at the expiration of (T_2^+, accept) and are increased upon the subsequent state transition to *accept*$^+$. Note that Statements (ii), (iii), (v), and (vii) trivially hold.

We now perform the induction step from $k \in \mathbb{N}$ to $k + 1$. Assume that Statements (ii) to (vii) hold for all values smaller or equal to k; Statement (i) only applies to $k = 1$ and was already shown. Define $l := \lfloor k/M \rfloor \geq 0$. Thus, if we can show Statement (ii) for $k + 1$, we may infer that

$$\begin{aligned} & t_{k+1} \\ & \overset{(i),(ii)}{\leq} t_{Ml} + (k + 1 - Ml)(T_1^+ + T_3^+ + \Sigma^+ + 5d_{\max}^+) + 3d \\ & \leq t_{Ml} + M(T_1^+ + T_3^+ + \Sigma^+ + 5d_{\max}^+) + 3d \qquad (42) \\ & \overset{(35)}{\leq} t_{Ml} + \frac{T_2 + T_4}{\vartheta} \,. \qquad (43) \end{aligned}$$

In case $l = 0$, it holds that $k < M$ and we may deduce (P1) by the same arguments as in the induction basis.

In case $l \geq 1$, we use Statement (v) for value k, and, by analogous arguments as in the induction basis, deduce that at no node in W, (T_2^+, accept) is expired during $[t_{Ml} + 3d, t_{Ml} + 3d + 3d_{\max}^+)$, implying that all nodes in W are in *accept*$^+$ during that time. Repeating the reasoning of the induction basis before (P1) with t_0 replaced by t_{Ml}, t_1 replaced by t_k, and t_M replaced by t_{Ml+M} shows that:

(P1') No node in W observes $(T_2^+, \mathit{accept})(t) = 0$ at some time $t \in [t_{Ml} + 3d, \min\{t_{Ml+M}, t_{Ml} + M(T_1^+ + T_3^+ + 5d_{\max}^+) + 3d\}]$ when it is not in state *accept*$^+$.

Since further $t_{Ml+M} \geq t_{k+1}$ by definition of l, we obtain from (P1') that no node $i \in W$ will memorize $\mathrm{NEXT}_i = 1$ earlier than time $\min\{t_{k+1}, t_{Ml} + M(T_1^+ + T_3^+ + 5d_{\max}^+) + 3d\}$ (again by reasoning analogously to the induction base).

By Statement (iv) for the value k, we know that each node $i \in W$ switches to *accept*$^+$ during $[t_k, t_k+\Sigma^+)$. In particular, i will increase its cycle counter at the respective time, i.e., Statement (vi) for $k+1$ follows at once if we establish Statement (vii) for $k+1$. As Statement (iv) for the value k together with Statement (ii) for value $k+1$ imply that each node in W switches to *accept*$^+$ exactly once during $[t_k, t_{k+1})$, Statement (vii) for $k+1$ follows, provided that we can exclude that the counter is reset to 0, due to (T_2^+, accept) expiring, at a time when it holds a non-zero value.

We now show that this never happens. By Statement (v) for value k each node $i \in W$ switches to *accept* during

$$[t_{Ml}, t_{Ml} + \Sigma^+ + 2d) \tag{44}$$

and this time is unique during $[t_{Ml}, t_{k+1})$ due to (42).

Because of (44) a node in W will reset its timeout (T_2^+, accept) during

$$[t_{Ml}, t_{Ml} + \Sigma^+ + 3d)\ , \tag{45}$$

and (T_2^+, accept) will expire within

$$\begin{aligned} &\left[t_{Ml} + \frac{T_2^+}{\vartheta}, t_{Ml} + T_2^+ + \Sigma^+ + 3d + d_{\max}^+\right) \\ \overset{(32)}{\subseteq}\ &\left[t_{Ml} + \frac{T_2^+}{\vartheta}, t_{Ml} + \frac{T_1^+}{\vartheta}\right) \\ \overset{(33)}{\subseteq}\ &[t_{Ml} + 3d + 3d_{\max}^+, t_{Ml+1})\ . \end{aligned}$$

Thus, no node in W leaves state *accept*$^+$ after switching there for the $(Ml)^{th}$ time after t_0+3d before observing that (T_2^+, accept) is reset and expires again. In particular, this shows that the counters are only reset to 0 at times when they are 0 anyway. Granted that Statement (ii) holds for $k+1$, Statement (vii) for $k+1$ follows.

Next, we establish Statements (ii) to (iv) for $k+1$. We reason analogously to the case of $k=1$, except that we have to revisit the conditions under which state *accept*$^+$ is left. As we have just seen, nodes in W switch from *accept*$^+$ to *ready*$^+$ upon T_1^+ expiring. Thus, as all nodes in W switch to *accept*$^+$ during $[t_k, t_k + \Sigma^+)$, they switch to *ready*$^+$ within the time window $[t_k + T_1^+/\vartheta, t_k + T_1^+ + \Sigma^+ + d_{\max}^+)$. By time

$$t_k + \frac{T_1^+}{\vartheta} \overset{(32)}{\geq} t_k + \Sigma^+ + d_{\max}^+\ ,$$

all nodes in W will be observed in *accept*$^+$ (and therefore not in *propose*$^+$), together with (P1') preventing that the first node in W that (directly) switches from *ready*$^+$ to *propose*$^+$ afterwards does so without T_3^+ expiring first.

More precisely, according to (P1') no node in W observes (T_2^+, accept) to be zero until time $\min\{t_{k+1}, t_{Ml} + M(T_1^+ + T_3^+ + 5d_{\max}^+) + 3d\}$. Further, each node $i \in W$ will be observed in state *propose*$^+$ no later than time $t_k + T_1^+ + T_3^+ + \Sigma^+ + 3d_{\max}^+$. As argued for $k=1$, it follows that indeed $t_{k+1} \leq t_k + T_1^+ + T_3^+ + \Sigma^+ + 3d_{\max}^+$, i.e., Statement (ii) for $k+1$ holds and thus Inequality (43). Further each node $i \in W$ switches to *accept*$^+$ for the $(k+1)^{th}$ time after $t_0 + 3d$ during time $[t_{k+1}, t_{k+1} + \Sigma^+)$, i.e., Statements (iv) for $k+1$ holds. Statement (iii) for $k+1$ is deduced from the fact that it takes at least $(T_1^+ + T_3^+)/\vartheta$ time until the first node from W switching to *propose*$^+$ after $t_k + \Sigma^+$ does so, since timeouts T_1^+ and T_3^+ need to be reset and expire first, one after the other.

Finally, we need to establish Statement (v) for $k+1$. If M does not divide $k+1$, Statement (v) for $k+1$ follows from Statement (v) for k. Otherwise M does divide $k+1$

and we can bound[32]

$$\begin{aligned} & t_{k+1} + \Sigma^+ + d \\ \overset{(i),(iii),(32)}{\geq} \quad & t_{Ml} + \frac{M(T_1^+ + T_3^+) - T_1^+ + T_2^+}{\vartheta} + \Sigma^+ + d \\ \overset{(35)}{\geq} \quad & t_{Ml} + T_2 + T_3 + \Sigma^+ + 4d \; . \end{aligned}$$

As by Statement (v) for k all nodes in W switched to *accept* during $[t_{Ml}, t_{Ml} + \Sigma^+ + 2d)$, we conclude[33] from the main state machines' description that all nodes in W are observed in state *ready* with timeout T_3 being expired (or already switched to *propose* or even *accept*) by time $t_{Ml} + T_2 + T_3 + \Sigma^+ + 4d$. Because all their NEXT signals switch to one during $[t_{k+1}, t_{k+1} + \Sigma^+)$, all nodes in W must therefore have switched to *propose* by time $t_{k+1} + \Sigma^+ + d$. Consequently, as we stated that w.l.o.g. By Inequality (43) they do not switch to *accept* again before time t_{k+1}. Consequently they do so at times in $[t_{k+1}, t_{k+1} + \Sigma^+ + 2d)$ as claimed.

This completes the induction. According to Statement (i), t_1 satisfies the claimed bound on the stabilization time. With respect to this time, Statement (iv) provides the skew bound, and combining it with Statements (ii) and (iii), respectively, yields the stated accuracy bounds. Statements (vi) and (vii) show the properties of the counters. Metastability-freedom of the state machine is trivially guaranteed by the fact that each state has a unique successor state. For the counter, we can infer metastability-freedom after stabilization from the observation made in the proof that for times $t \geq t_1$, $(T_2, \mathit{accept})(t) = 0$ at a non-faulty node implies that it is in state accept^+ with its cycle counter equal to zero. This completes the proof. □

For some applications, one might require an even higher operational frequency than provided by the quick cycle state machine. It turns out that there is a simple solution to this issue.

Increasing the Frequency Further

Given any pulse synchronization protocol, one can derive clocks operating at an arbitrarily large frequency as follows. Whenever a pulse is triggered locally, the nodes start to increase a local integer counter modulo some value $m \in \mathbb{N}$ at a speed of $\phi \in \mathbb{R}^+$ times that of a local clock, starting from 0. Denote by T^- the accuracy lower bound of the protocol and suppose that the local clock controlling the counter runs at a speed between 1 and $\rho \in (1, \vartheta]$, i.e., its maximum drift is $\rho - 1$.[34] Once the counter reaches the value $m - 1$, it is halted until the next pulse. We demand that

$$m \leq \phi T^- \; . \tag{46}$$

Note that although the clock gets halted, the amortized frequency is essentially optimal. The maximal time the clock is halted is proportional to $\rho - 1$ multiplied by the time it takes to complete an iteration of the quick cycle; thus, halting the clock should be uncritical except for applications requiring extremely fast real-time response.

This approach is similar to the one presented in [Daliot and Dolev 2006], enriched by addressing the problem of metastability. In the context of the FATAL$^+$ protocol, we get the following result.

[32] Note that we already build on Statement (iii) for $k + 1$ here.

[33] This statement relies on the constraints on the main state machines' timeouts, which require that T_2 expiring is the critical condition for switching to *ready*.

[34] We introduce ρ since one might want to invest into a single, more accurate clock source per node in order to obtain smaller skews.

COROLLARY 6.3. *Adding a counter as described above to the FATAL$^+$ protocol and concatenating the counter values of the two counters at node $i \in V$ yields a bounded logical clock $L_i \in \{0, \dots, mM-1\}$. At any time t when the protocol has stabilized on some set W (according to Theorem 6.2), it holds for any two nodes $i, j \in W$ that*

$$|L_i(t) - L_j(t) \bmod mM| \leq \left\lceil \phi\Sigma^+ + \left(1 - \frac{1}{\rho}\right) m \right\rceil .$$

Once stabilized, these clocks do not "jump", i.e., they always increase by exactly one mod mM, with at least $1/\rho$ time between any two consecutive "ticks".

The amortized clock frequency is within the bounds $m/(T_1^+ + T_3^+ + 2\Sigma^+ + 3d_{\max}^+)$ and $\vartheta m/(T_1^+ + T_3^+ - \Sigma^+)$. Viewed as a state machine in our model, the clocks L_i, where $i \in W$, are metastability-free after stabilization.

PROOF. Observe that it takes at least $m/(\phi\rho)$ time for one of the new counters to increase from 0 to m. Since the counters are restarted at pulses, which are triggered locally at most Σ^+ time apart, at the time when a "fast" node arrives at the value m, a "slow" node will have increased its clock by at least $\lfloor m/\rho - \phi\Sigma^+ \rfloor$. According to Inequality (46), slow nodes will be able to increase their counters to m before the next pulse. The claimed bound on the clock skew and the facts that clock increases are one by one and at most every $1/\rho$ time follow.

The bound on the amortized clock frequency follows by considering the minimal and maximal times M iterations of the quick cycle may require.

The metastability-freedom of the clock is deduced from the metastability-freedom of the individual counters. For the new counter this is guaranteed by Inequality (46), since the counter is always halted at 0 before it is reset due to a new quick cycle pulse. □

We remark that in an implementation, one would probably utilize the better clock source, if available, to drive T_1^+ and T_3^+ as well.[35] Maximizing m with respect to Inequality (46) and choosing $T_1^+ + T_3^+$ sufficiently large will thus result in clocks whose amortized drift is arbitrarily close to ρ, the drift of the underlying local clock source.

7. DISCUSSION

In this work, we presented a novel Byzantine tolerant self-stabilizing pulse synchronization algorithm. Our analysis shows that the algorithm is optimal in several aspects. It tolerates up to $f < n/3$ faults, deals with worst-case clock drifts and delays, requires channels of constant bandwidth only, is metastability-free after stabilization in fault-free runs, and achieves constant-time stabilization of nodes that (re)join the operational system.

Moreover, as shown in detail in Dolev et al. [2014], the algorithm is suited for implementation in hardware. In particular, the complexity of nodes is small as well. Nodes need to evaluate constant-bit signals from $\mathcal{O}(n)$ channels and comprise a linear number of timeouts and clocks. With the exception of determining whether a threshold of $f+1$ or $n-f$ signals is reached, local computations require a linear number of gates only (cf. Függer et al. [2006]). Checking for thresholds of asynchronous, monotonically increasing signals in a metastability-free manner can be done by sorting networks of depth $\mathcal{O}(\log n)$ and $\mathcal{O}(n \log n)$ gates [Ajtai et al. 1983].[36] Clearly, it is necessary to have

[35] Since the new counter is started together with T_1^+, this does not incur metastability. Special handling is required for T_3^+ on the M^{th} pulse of the quick cycle, though.

[36] The constants are extremely large, implying that for smaller values of n different solutions with asymptotic depth $\mathcal{O}(\log^2 n)$, but good constants, are preferred. For very small n, e.g. 4 ($f=1$) or 7 ($f=2$), different approaches of exponential asymptotic complexity can be of interest as well.

conditions involving more than f nodes in order to recover despite f Byzantine faulty nodes. Therefore, for gates of constant fan-in, our solution is trivially optimal in terms of the delay arising from local computations, and nodes' gate complexity is at most a factor of $\mathcal{O}(\log n)$ from the optimum.

Whether the complexity of our algorithm in terms of stabilization time or the number of bits transmitted during the stabilization process is optimal remains open. There are quite a few structural similarities to consensus and agreement problems, though. In fact, some existing pulse synchronization algorithms directly utilize consensus algorithms as subroutine [Ben-Or et al. 2008; Daliot and Dolev 2006]. However, in general consensus lower bounds do not directly apply to our setting, as nodes need to achieve *approximate* agreement on when to fire pulses only. For this problem, no nontrivial lower bounds are known. Nonetheless, lower bounds for consensus might be an indication for the complexity of the problem, as—in contrast to the synchronous setting of Ben-Or et al. [2008]—in a bounded-delay system achieving approximate agreement on a time is necessary to enable simulating a synchronous protocol. Therefore, although approximate agreement is known to be easier than consensus in synchronous systems [Dolev et al. 1986], one might conjecture that the randomized time complexity of achieving self-stabilizing pulse synchronization in a bounded-delay system with the considered adversary is the same as for randomized consensus in an asynchronous system with a strong adversary. If this was indeed true, our algorithm would be optimal with respect to stabilization time and for stabilization with probability $1 - 2^{-\Omega(n)}$ [Attiya and Censor 2008].

Apart from asymptotic considerations, it is of practical concern what the additive and multiplicative constants in the asymptotic complexities are. While the bandwidth required between any two nodes is a mere five bits and the number of components per node is fairly low,[37] the stabilization time suffers from a large multiplicative constant. For the envisioned application, this is however not critical: Overall system failure should be a rare event, and the high operational frequencies desired in practice will help to keep the real-time required for stabilization low. This is supported by results of an evaluation of an initial FPGA prototype, showing stabilization times of no more than 10 seconds in over 250,000 test runs for $d \approx 5\,\mu s$ in an 8 node system [Dolev et al. 2014].[38] Systems operating at today's typical clock speeds in the gigahertz range will have a maximum delay—and thus stabilization time—that is roughly by three orders of magnitude smaller. Moreover, as long as the majority of nodes continues to operate within specifications, a node subject to a transient fault will stabilize deterministically, by about another two orders of magnitude faster (cf. Theorem 4.17). Note that this bound, which is independent of n, is the one determining the mean time until failure of the system as a whole.

Despite all advantageous properties and the promising results of the test implementation, there remains an obstacle to putting the algorithm to use. The fact that the nodes need to be fully connected for our pulse synchronization routine (as is the case for all current Byzantine-tolerant self-stabilizing clock synchronization algorithms) constitutes a major scalability issue. Sadly, tolerating a constant fraction of Byzantine faulty nodes comes at the cost of linear node connectivity in the worst case [Dolev 1982; Fischer et al. 1985]. However, if one assumes a (spatially) independent and uniform distribution of faults, it becomes feasible to partition the system in cliques of size

[37] Nodes never need to evaluate more than three threshold conditions concurrently and the remaining logic is simple in comparison.

[38] These values match the bounds from theory, which however is not very surprising given that these determine the values picked for the timeouts. For practical purposes, the constraints on the timeouts derived from the analysis are probably overly conservative in most cases.

$\mathcal{O}(\log n)$ that stay operational with high probability granted that the probability of individual nodes being faulty is at most constant. These cliques then may be connected by a constant-degree graph (where each link is emulated by fully connecting the respective cliques), enabling to emulate an efficient clock synchronization algorithm that is not Byzantine-tolerant, but in turn runs on arbitrary graphs. As the worst-case skew must grow linearly in the diameter of a graph [Biaz and Welch 2001], a particularly attractive option here is gradient clock synchronization [Kuhn et al. 2010; Lenzen et al. 2010]. The goal of gradient clock synchronization algorithms is, for any pair of nodes, to achieve a tighter synchronization the closer the respective nodes are.

In the resulting system, node degree, gate complexity, and (computational) delay would drop to $\mathcal{O}(\log n)$, $\mathcal{O}(\log n \log\log n)$, and $\mathcal{O}(\log\log n)$, respectively. Note that stabilization of all $\Omega(n/\log n)$ cliques will merely require $\mathcal{O}(\log n)$ time with high probability due to the large probability of stabilization of individual cliques. Even a pulse synchronization algorithm with a constant probability of stabilization within $\mathcal{O}(1)$ time would not perform better in this setting. Implementing these ideas is subject to future work, and hopefully will lead to a clocking scheme for large-scale Systems-on-Chip that is of great efficiency, yet offers high resilience to faults.

REFERENCES

AJTAI, M., KOMLÓS, J., AND SZEMERÉDI, E. 1983. An $\mathcal{O}(n \log n)$ Sorting Network. In *Proc. 15th Symposium on Theory of Computing (STOC)*. 1–9.

ATTIYA, H. AND CENSOR, K. 2008. Lower Bounds for Randomized Consensus Under a Weak Adversary. In *Proc. 27th Symposium on Principles of Distributed Computing (PODC)*. 315–324.

BEN-OR, M., DOLEV, D., AND HOCH, E. N. 2008. Fast Self-Stabilizing Byzantine Tolerant Digital Clock Synchronization. In *Proc. 27th Symposium on Principles of Distributed Computing (PODC)*. 385–394.

BHAMIDIPATI, R., ZAIDI, A., MAKINENI, S., LOW, K., CHEN, R., LIU, K.-Y., AND DALGREHN, J. 2002. Challenges and Methodologies for Implementing High-Performance Network Processors. *Intel Technology Journal 6,* 3, 83–92.

BIAZ, S. AND WELCH, J. 2001. Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions. *Information Processing Letters 80,* 3, 151–157.

CHAPIRO, D. M. 1984. Globally-Asynchronous Locally-Synchronous Systems. Ph.D. thesis, Stanford University.

CONSTANTINESCU, C. 2003. Trends and Challenges in VLSI Circuit Reliability. *IEEE Micro 23,* 4, 14–19.

DALIOT, A. AND DOLEV, D. 2006. Self-Stabilizing Byzantine Pulse Synchronization. *Computing Research Repository abs/cs/0608092*.

DALIOT, A., DOLEV, D., AND PARNAS, H. 2003. Self-Stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks. In *Proc. 6th Symposium on Self-Stabilizing Systems (SSS)*.

DIKE, C. AND BURTON, E. 1999. Miller and Noise Effects in a Synchronizing Flip-Flop. *IEEE Journal of Solid-State Circuits SC-34,* 6, 849–855.

DOLEV, D. 1982. The Byzantine Generals Strike Again. *Journal of Algorithms 3*, 14–30.

DOLEV, D., FUEGGER, M., LENZEN, C., POSCH, M., SCHMID, U., AND STEININGER, A. 2014. Rigorously modeling self-stabilizing fault-tolerant circuits: An ultra-robust clocking scheme for systems-on-chip. *Journal of Computer and System Sciences*. To appear.

DOLEV, D. AND HOCH, E. 2007. Byzantine Self-Stabilizing Pulse in a Bounded-Delay Model. In *Proc. 9th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 4280. 350–362.

DOLEV, D., LYNCH, N. A., PINTER, S. S., STARK, E. W., AND WEIHL, W. E. 1986. Reaching Approximate Agreement in the Presence of Faults. *Journal of the ACM 33*, 499–516.

DOLEV, S. AND WELCH, J. L. 2004. Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults. *Journal of the ACM 51,* 5, 780–799.

FISCHER, M. J. AND LYNCH, N. A. 1982. A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters 14*, 183–186.

FISCHER, M. J., LYNCH, N. A., AND MERRITT, M. 1985. Easy Impossibility Proofs for Distributed Consensus Problems. In *Proc. 4th Conference of Principles of Distributed Computing (PODC)*. 59–70.

FRIEDMAN, E. G. 2001. Clock Distribution Networks in Synchronous Digital Integrated Circuits. *Proceedings of the IEEE 89,* 5, 665–692.

FUCHS, G., FÜGGER, M., AND STEININGER, A. 2009. On the Threat of Metastability in an Asynchronous Fault-Tolerant Clock Generation Scheme. In *Proc. 15th Symposium on Asynchronous Circuits and Systems (ASYNC)*. Chapel Hill, N. Carolina, USA, 127–136.

FÜGGER, M. 2010. Analysis of On-Chip Fault-Tolerant Distributed Algorithms. Ph.D. thesis, Technische Universität Wien, Institut für Technische Informatik.

FÜGGER, M., DIELACHER, A., AND SCHMID, U. 2010. How to Speed-Up Fault-Tolerant Clock Generation in VLSI Systems-on-Chip via Pipelining. In *Proc. 8th European Dependable Computing Conference (EDCC)*. 230–239.

FÜGGER, M. AND SCHMID, U. 2012. Reconciling Fault-Tolerant Distributed Computing and Systems-on-Chip. *Distributed Computing 24,* 6, 323–355.

FÜGGER, M., SCHMID, U., FUCHS, G., AND KEMPF, G. 2006. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In *Proc. 6th European Dependable Computing Conference (EDCC)*. 87–96.

GADLAGE, M. J., EATON, P. H., BENEDETTO, J. M., CARTS, M., ZHU, V., AND TURFLINGER, T. L. 2006. Digital Device Error Rate Trends in Advanced CMOS Technologies. *IEEE Transactions on Nuclear Science 53,* 6, 3466–3471.

HOCH, E., DOLEV, D., AND DALIOT, A. 2006. Self-Stabilizing Byzantine Digital Clock Synchronization. In *Proc. 8th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2006)*. Vol. 4280. 350–362.

International Technology Roadmap for Semiconductors 2012. International Technology Roadmap for Semiconductors. http://www.itrs.net.

KINNIMENT, D. J., BYSTROV, A., AND YAKOVLEV, A. V. 2002. Synchronization Circuit Performance. *IEEE Journal of Solid-State Circuits SC-37,* 2, 202–209.

KUHN, F., LENZEN, C., LOCHER, T., AND OSHMAN, R. 2010. Optimal Gradient Clock Synchronization in Dynamic Networks. In *29th Symposium on Principles of Distributed Computing (PODC)*.

LENZEN, C., LOCHER, T., AND WATTENHOFER, R. 2010. Tight Bounds for Clock Synchronization. In *Journal of the ACM*. Vol. 57(2).

LUNDELIUS, J. AND LYNCH, N. 1984. An Upper and Lower Bound for Clock Synchronization. *Information and Control 62,* 2-3, 190–204.

MALEKPOUR, M. 2006. A Byzantine-Fault Tolerant Self-stabilizing Protocol for Distributed Clock Synchronization Systems. In *Proc. 9th Conference on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 411–427.

MALEKPOUR, M. 2009. A Self-Stabilizing Byzantine-Fault-Tolerant Clock Synchronization Protocol. Tech. rep., NASA. TM-2009-215758.

MARINO, L. 1981. General Theory of Metastable Operation. *IEEE Transactions on Computers C-30,* 2, 107–115.

METRA, C., FRANCESCANTONIO, S., AND MAK, T. 2004. Implications of Clock Distribution Faults and Issues with Screening them During Manufacturing Testing. *IEEE Transactions on Computers 53,* 5, 531–546.

PEASE, M., SHOSTAK, R., AND LAMPORT, L. 1980. Reaching Agreement in the Presence of Faults. *Journal of the ACM 27*, 228–234.

POLZER, T., HANDL, T., AND STEININGER, A. 2009. A Metastability-Free Multi-Synchronous Communication Scheme for SoCs. In *Proc. 11th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 578–592.

PORTMANN, C. L. AND MENG, T. H. Y. 1995. Supply Noise and CMOS Synchronization Errors. *IEEE Journal of Solid-State Circuits SC-30,* 9, 1015–1017.

RESTLE, P., MCNAMARA, T., WEBBER, D., CAMPORESE, P., ENG, K., JENKINS, K., ALLEN, D., ROHN, M., QUARANTA, M., BOERSTLER, D., ALPERT, C., CARTER, C., BAILEY, R., PETROVICK, J., KRAUTER, B., AND MCCREDIE, B. 2001. A Clock Distribution Network for Microprocessors. *IEEE Journal of Solid-State Circuits 36,* 5, 792–799.

SEMIAT, Y. AND GINOSAR, R. 2003. Timing Measurements of Synchronization Circuits. In *Proc. 9th Symposium on Asynchronous Circuits and Systems (ASYNC)*.

SRIKANTH, T. K. AND TOUEG, S. 1987. Optimal Clock Synchronization. *Journal of the ACM 34,* 3, 626–645.

SUNDARESAN, K., ALLEN, P., AND AYAZI, F. 2006. Process and Temperature Compensation in a 7-MHz CMOS Clock Oscillator. *IEEE J. Solid-State Circuits 41,* 2, 433–442.

TEEHAN, P., GREENSTREET, M., AND LEMIEUX, G. 2007. A Survey and Taxonomy of GALS Design Styles. *IEEE Design and Test of Computers 24,* 5, 418–428.

WIDDER, J. AND SCHMID, U. 2009. The Theta-Model: Achieving Synchrony without Clocks. *Distributed Computing 22,* 1, 29–47.