On Self-stabilizing Synchronous Actions Despite Byzantine Attacks*

Danny Dolev** and Ezra N. Hoch

School of Engineering and Computer Science,
The Hebrew University of Jerusalem, Israel,
{dolev,ezraho}@cs.huji.ac.il

Abstract. Consider a distributed network of n nodes that is connected to a global source of "beats". All nodes receive the "beats" simultaneously, and operate in lock-step. A scheme that produces a "pulse" every *Cycle* beats is shown. That is, the nodes agree on "special beats", which are spaced *Cycle* beats apart. Given such a scheme, a clock synchronization algorithm is built. The "pulsing" scheme is self-stabilized despite any transient faults and the continuous presence of up to $f < \frac{n}{3}$ *Byzantine* nodes. Therefore, the clock synchronization built on top of the "pulse" is highly fault tolerant. In addition, a highly fault tolerant general stabilizer algorithm is constructed on top of the "pulse" mechanism.
Previous clock synchronization solutions, operating in the exact same model as this one, either support $f < \frac{n}{4}$ and converge in linear time, or support $f < \frac{n}{3}$ and have **exponential convergence time** that also depends on the value of *max-clock* (the clock wrap around value). The proposed scheme combines the best of both worlds: it converges in **linear time** that is independent of *max-clock* and is tolerant to up to $f < \frac{n}{3}$ *Byzantine* nodes. Moreover, considering problems in a self-stabilizing, *Byzantine* tolerant environment that require nodes to know the global state (clock synchronization, token circulation, agreement, etc.), the work presented here is the first protocol to operate in a network that is not fully connected.

1 Introduction

Most distributed tasks require some sort of synchronization. Clock synchronization is a very basic and intuitive tool for supplying this. PULSE synchronization can be used as an underlying building block to achieve clock synchronization, as well as solving other synchronization problems; in a sense, PULSE synchronization is a more fundamental synchronization problem.

It thus makes sense to require an underlying PULSE synchronization mechanism to be highly fault-tolerant. This paper presents a PULSE synchronization algorithm that is self-stabilizing and is tolerant to permanent presence of

* A pre-copy of the paper appearing in Disc, Sept. 2007.
** Part of the work was done while the author visited Cornell university. The work was funded in part by ISF, ISOC, NSF, CCR, and AFOSR.

Byzantine faults. That is, it attains synchronization, once lost, while containing the influence of the permanent presence of faulty nodes.

Consider a system in which the nodes execute in lock-step by regularly receiving a common "pulse" or "tick" or "beat". The objective is to agree on some "special beats" that are $Cycle$ beats apart. We will use the "beat" notation for the "global" signal received, and "pulse" for the "special beats" agreed upon.

The PULSE synchronization problem is to ensure that eventually all correct nodes PULSE together, and as long as enough nodes remain correct, they continue to PULSE together, $Cycle$ beats apart. For example, given $Cycle = 7$ we would like all correct nodes, that may start at arbitrary initial states, to eventually PULSE together every 7 beats, and continue so as long as there are enough correct nodes.

The global beat system provides some measure of synchronization. For example, given a global beat system with beat interval at least as long as the worst-case execution-time for terminating *Byzantine* agreement, the PULSE synchronization problem is solved by initiating a *Byzantine* agreement on the next time when the nodes should PULSE, each time a beat is received. The crux of the problem is to achieve synchronization when it is not given by the global beat system; that is, when the beat interval length is in the order of the communication's end-to-end delay. Since in that scenario the global beat system does not provide - by itself - enough synchronization, and a more complex algorithm is required to exert the required synchronization. The main contribution of the current paper is achieving exactly that.

Related Work: PULSEing has been used as an underlying fault tolerant mechanism in clock synchronization, token circulation and to create a general stabilizer (see [4] for an overview). All of these algorithms are self-stabilizing and *Byzantine* tolerant, due to the fault tolerant nature of the underlying PULSE mechanism. This gives the motivation for producing robust and efficient PULSEing algorithms, as they can be used to improve the robustness of a variety of applications.

Clock synchronization is one of the first problems that was solved in a self-stabilizing and *Byzantine* tolerant fashion. In [9] and [12] it was solved directly, and in [4] it was solved using an underlying PULSEing algorithm. [9] was the first work to discuss the exact same model as presented here, as opposed to [4], which operates without a global beat system.

Synchronization of clocks of integer values was previously termed digital clock synchronization ([2,7,8,16]) or "synchronization of phase-clocks" ([11]). However, in this paper we concentrate on the PULSEing mechanism, as it yields clock synchronization as well as other fault tolerant protocols.

Several fault tolerant stabilizers exist (see [1], [10] and [13]) with varying requirement and features (such as local containment of faults). In [3], it was shown that PULSE synchronization can be used to create a generalized stabilizer. However, in [3] the stabilizer is complex, and can stabilize a narrow class of algorithms. In Section 9 we show a simpler stabilizer, which can stabilize a wider range of algorithms.

Some of the previous results combining *Byzantine* faults and self-stabilization consider a class of problems in which the state of each correct node is determined locally. Usually such solutions can operate in a general graph (see [17], [15] and [14]) without the need to aggregate or accumulate information across the network. In the class of problems in which the state of each correct node is correlated with the state of the other correct nodes, the current paper is the first paper to present a solution that operates in a network that is not fully connected.

Contributions: We construct a self stabilizing PULSEing algorithm, that is tolerant to up to $f < \frac{n}{3}$ *Byzantine* nodes, and converges in linear time, for any target interval of pulsing.

As will be shown in Section 8, clock synchronization and PULSE synchronization are equivalent. Hence, this work is compared to the state of the art of previous clock synchronization results that operate in the exact same model.

Previous results have either linear convergence time with $f < \frac{n}{4}$ (see [12]) or exponential convergence time with $f < \frac{n}{3}$ (see [9]). In this paper we obtain a linear convergence time with $f < \frac{n}{3}$. Moreover, our convergence time is independent of the *max-clock* value (the clock wrap around value) of the digital clock, in contrast to [9].

In addition, our algorithm is the first one in this model and for this type of problems that does not require each node to be connected to every other node, it only requires that there are $2 \cdot f + 1$ distinct routes between any two correct nodes, matching the lower bound of [5].

2 Model and Definitions

Consider a fully connected network of n nodes (we later generalize the results to a more general network). All the nodes are assumed to have access to a "global beat system" that provides "beats" with regular intervals. The communication network and all the nodes may be subject to severe transient failures, which might leave the system in an arbitrary state.

We say that a node is *Byzantine* if it does not follow the instructed algorithm and *non-Byzantine* otherwise. Thus, any node whose failure does not allow it to exactly follow the algorithm as instructed is considered *Byzantine*, even if it does not behave fully maliciously. A non-*Byzantine* node will be called *non-faulty*. In the following discussion f will denote the upper bound on the number of *Byzantine* nodes.[1] The presented solution supports $f < \frac{n}{3}$.

Definition 1. *The system is* coherent *if there are at most f Byzantine nodes, and each message sent at a beat to a non-faulty destination arrives and is processed at its destination before the next beat.*

Nodes are instructed to send their messages immediately after the occurrence of a beat from the global beat system. Therefore, when the system is coherent

[1] In the literature the term "permanent" *Byzantine* node is sometimes used.

message delivery and the processing involved can be completed between two consecutive global beats, by any node that is non-faulty. More specifically, the time required for message delivery and message processing is called a *round*, and we assume that the time interval between global beats is greater than and in the order of such a round. Due to transient faults, different nodes might not agree on the current beat/round number. We will use the notion of an external beat number r, which the nodes are not aware of, but will simplify the proofs' presentation and discussion.

At times of transient failures there can be any number of concurrent *Byzantine* faulty nodes; the turnover rate between faulty and non-faulty behavior of nodes can be arbitrarily large and the communication network may also behave arbitrarily. Eventually the system behaves coherently again. At such case a non-faulty node may still find itself in an arbitrary state. Since a non-faulty node may find itself in an arbitrary state, there should be some time of continues non-faulty operation before it can be considered correct.

Definition 2. *A non-faulty node is considered* correct *only if it remains non-faulty for Δ_{node} rounds during which the system is coherent.*[2]

The algorithm parameters n, f, as well as the node's id are fixed constants and thus are considered part of the incorruptible correct code at the node. Thus, it is assumed that non-faulty nodes do not hold arbitrary values of these constants.

2.1 The PULSEing Problem

We say that a system is $[\phi, \psi]$-PULSING if all correct nodes PULSE together in the following pattern: ϕ consecutive beats of PULSEs followed by ψ consecutive beats of non-PULSE. That is, the system has a *Cycle* of length $\phi+\psi$ beats, out of which only the first ϕ beats are PULSEs. More formally, denote by $pulsed_p(r) = True$ if p PULSEd on beat r and $pulsed_p(r) = False$, otherwise.

Definition 3. *A system is $[\phi, \psi]$-PULSING in the beat interval $[r_1, r_2]$ if there exists some $0 \leq k < \phi + \psi$, such that for every correct node p, and for every beat $r \in [r_1, r_2]$, it holds that:*

1. *$pulsed_p(r) = True$, in case $0 \leq r - k \ (mod\ \phi + \psi) < \phi$; and*
2. *$pulsed_p(r) = False$ in case $\phi \leq r - k \ (mod\ \phi + \psi) < \phi + \psi$.*

(k denotes the offset, from r_1, of the first PULSE in the pattern.)

For example, consider "1" to represent a beat in which all correct nodes PULSE, and "0" a beat in which all correct nodes do not PULSE. Using this notation, the following is a PULSEing pattern of a $[\phi, \psi]$-PULSING system.

$$[\ldots, \overbrace{\underbrace{1, 1, \ldots, 1}_{\phi \text{ beats}}, \underbrace{0, 0, \ldots, 0}_{\psi \text{ beats}}}^{Cycle \text{ beats}}, \overbrace{\underbrace{1, 1, \ldots, 1}_{\phi \text{ beats}}, \underbrace{0, 0, \ldots, 0}_{\psi \text{ beats}}}^{Cycle \text{ beats}}, \ldots]$$

[2] The assumed bound on the value of Δ_{node} is defined in Remark 3.

Definition 4. The PULSE**ing problem:**
Convergence: *Starting from an arbitrary state, the system becomes* $[\phi, \psi]$-PULSING *after a finite number of beats.*
Closure: *If the system is* $[\phi, \psi]$-PULSING *in the beat interval* $[r_1, r_2]$ *it is also* $[\phi, \psi]$-PULSING *in the interval* $[r_1, r_2 + 1]$.

Definition 5. *a* $[\phi, \psi]$-PULSER *is an algorithm* $\mathcal{A}$*, such that once the system is coherent (and stays so), it solves the* PULSE*ing problem.*

The objective is to develop an algorithm that PULSEs only once every $Cycle$.

Notation: We denote a $[1, \psi]$-PULSER as $[\psi + 1]$-PULSER.

Using the previously used notation of "1" for PULSEing and "0" for non-PULSEing, a $[Cycle]$-PULSER looks as follows:

$$[\ldots, \underbrace{1, 0, 0, \ldots, 0}_{Cycle \text{ beats}}, \underbrace{1, 0, 0, \ldots, 0}_{Cycle \text{ beats}}, \ldots]$$

The goal is to build a $[Cycle]$-PULSER for any $Cycle > 0$. That is, a self-stabilizing, *Byzantine* tolerant algorithm that eventually PULSEs every $Cycle$ beats. The following section outlines the solution.

3 Constructing a $[Cycle]$-PULSER

In contrast to previous solutions that were very involved, the new solution presented below is more modular. Addressing the problem in a modular way enabled us to unwrap the difficulties in solving the problem, and to come up with a tight solution. Its modularity also enables to simplify the proof of correctness and to better present the intuition behind it. The core of the protocol is the Large-Cycle-Pulser algorithm that produces a $[\Delta, \Delta + Cycle']$-PULSER. This module uses another module called $\mathcal{BBB}$ to limit the ability of the *Byzantine* nodes to disrupt the protocol. To obtain the complete solution the core protocol is wrapped with two additional modules, as detailed below.

We first show how to construct a $[\Delta, \Delta + Cycle']$-PULSER $\mathcal{A}$ for any $Cycle' > \Delta$, where Δ is a bound on running a given distributed agreement protocol. We continue by showing how to construct a $[\phi + \psi]$-PULSER from any $[\phi, \psi]$-PULSER. Using this, we construct a $[2 \cdot \Delta + Cycle']$-PULSER $\mathcal{A}'$ for any $Cycle' > \Delta$. Lastly, using $\mathcal{A}'$, we construct a $[Cycle]$-PULSER for any $Cycle \geq 1$.

Remark 1. Note that $[\phi + \psi]$-PULSER is actually $[1, \phi + \psi - 1]$-PULSER, and hence a $[\phi, \psi]$-PULSER (pulses for ϕ beats then it is quiet for ψ beats) is transformed into a $[1, \phi + \psi - 1]$-PULSER (PULSEs once, then it is quiet for $\phi + \psi - 1$ beats).

The construction of $\mathcal{A}$ uses a building block that is essentially a *Byzantine* consensus. We denote this building block by $\mathcal{BBB}$ (*Byzantine* Black Box).

3.1 The *Byzantine* Black Box Construction

$\mathcal{BBB}$ is defined to be a round based distributed protocol, such that each node p has a binary input value v_p and a binary output value $\mathcal{V}_p$. $\mathcal{BBB}$ has the following properties:

1. *Termination*: The algorithm terminates within Δ rounds.
2. *Agreement*: All *non-faulty*[3] nodes agree on the same output value $\mathcal{V}$. That is, for any two *non-faulty* nodes p, p' it holds that $\mathcal{V}_p = \mathcal{V}_{p'} = \mathcal{V}$.
3. *Validity*: If $n - f$ *non-faulty* nodes have the same input value ν then that is the output value, $\mathcal{V} = \nu$.

$\mathcal{BBB}$ is required to be *Byzantine* tolerant, but is not required to be self-stabilizing. The self-stabilization of the $[\phi + \psi]$-PULSER $\mathcal{A}$ (presented later) will not be hampered by this.[4] In addition, $\mathcal{A}$ will rely only on the properties of $\mathcal{BBB}$ (when it is executed by enough correct nodes) for its operation. In $\mathcal{A}$ all messages exchanged among the nodes will use $\mathcal{BBB}$. Since the presented $\mathcal{BBB}$ can tolerate $f < \frac{n}{3}$ faulty nodes, $\mathcal{A}$ can tolerate the same failure ratio.

Remark 2. $\mathcal{BBB}$ can be implemented via any algorithm that solves the *Byzantine* consensus problem; the only difference lies in the "validity" condition, where instead of limiting the validity to the case that "all correct nodes" start with the same initial value, $\mathcal{BBB}$ limits the validity condition to having only $n - f$ non-faulty nodes with the same initial value, even if there happen to be more non-faulty nodes at that instance.

Remark 3. A non-faulty node that has recently recovered from a transient failure cannot immediately be considered correct. In the context of this paper, a non-faulty node is considered *correct* once it remains non-faulty for at least $\Delta_{node} = \Delta + 1$, and as long as it continues to be non-faulty.

3.2 A $[\Delta, \Delta + Cycle']$-PULSER

Figure 1 presents an algorithm that produces a $[\Delta, \Delta + Cycle']$-PULSER, for $Cycle' > \Delta$. This algorithm executes Δ simultaneous $\mathcal{BBB}$ protocols. Consider $\mathcal{BBB}_i$ as a "pointer" to a $\mathcal{BBB}$ instance, hence the statement

$$\mathcal{BBB}_2 := \mathcal{BBB}_1; \mathcal{BBB}_1 := new\ \mathcal{BBB}(\text{“1”});$$

means that $\mathcal{BBB}_2$ will contain the previous instance of $\mathcal{BBB}_1$, and $\mathcal{BBB}_1$ will contain a new instance of $\mathcal{BBB}$ initialized with the input value 1. The output value of $\mathcal{BBB}_i$ is $\mathcal{V}(\mathcal{BBB}_i)$.

[3] in the context of $\mathcal{BBB}$, a node is considered non-faulty only if it is non-faulty throughout the whole execution of $\mathcal{BBB}$.

[4] $\mathcal{BBB}$ is initiated, executed and terminated repeatedly; each instance starts with a "clean slate", thus not harming the self-stability of the algorithm that uses it.

Algorithm **Large-Cycle-Pulser** */* executed repeatedly at each beat */*

1. for each $i \in \{1, .., \Delta\}$ do
   execute the i^{th} round of the $\mathcal{BBB}_i$ protocol;
2. (a) if $Counter > 0$ then
   $Counter := \min\{Counter - 1, Cycle'\}$;
   $WantToPulse := 0$;
   (b) else
   $WantToPulse := 1$;
3. if $\mathcal{V}(\mathcal{BBB}_\Delta) = 1$ then
   (a) do PULSE;
   (b) $Counter := Cycle'$;
4. for each $i \in \{2, ..., \Delta\}$ do
   $\mathcal{BBB}_i := \mathcal{BBB}_{i-1}$;
5. initialize a new instance of $\mathcal{BBB}$, $\mathcal{BBB}_1 = \mathcal{BBB}(WantToPulse)$.

Fig. 1. A $[\Delta, \Delta + Cycle']$-PULSER algorithm for $Cycle' > \Delta$.

4 Proof of Large-Cycle-Pulser's correctness

All the lemmata, theorems, corollaries and definitions hold only as long as the system is coherent. We assume that nodes may start in an arbitrary state, and nodes may fail and recover, but from some time on, at any round there are at least $n - f$ correct nodes.

Let $\mathcal{G}$ denote a group of non-*Byzantine* nodes that behave according to the algorithm, and that are not subject to (for some pre-specified number of rounds) any new transient failures. We will prove that if $|\mathcal{G}| \geq n - f$ and all of these nodes remain non-faulty for a long enough period of time ($\Omega(\Delta)$ global beats), then the system will converge.

For simplifying the notations, the proofs refer to some "external" beat number. The nodes do not maintain it and have no access to it; it is only used for the proofs' clarity.

Definition 6. *A group $\mathcal{G}$ is* CORRECT(α, β) *if $|\mathcal{G}| \geq n-f$, and every node $p \in \mathcal{G}$ is correct during the beat interval $[\alpha, \beta]$. Let δ mark the length of the interval, that is $\delta = \beta - \alpha + 1$.*

Note that each node $p \in \mathcal{G}$, when $\mathcal{G}$ is CORRECT(α, β), has not been subject to a transient failure in the beat interval $[\alpha - \Delta_{node}, \beta]$; and is non-faulty during that interval.

Definition 7. *We say that a system is* CORRECT(α, β) *if there exists a set $\mathcal{G}$ such that $\mathcal{G}$ is* CORRECT(α, β).

In the following lemmata, $\mathcal{G}$ refers to any set implied by CORRECT(α, β), without stating so specifically. The proofs hold for any such set $\mathcal{G}$.

Note that if the system is coherent, and there has not been a transient failure for at least $\Delta + 1$ beats, then, by definition, $\mathcal{G}$ contains all nodes that were non-faulty during that period.

Lemma 1. *$\forall\beta \geq \alpha$: If the system is* CORRECT(α, β) *then at any beat $r \in [\alpha, \beta]$, either all nodes in $\mathcal{G}$* PULSE *or they all do not* PULSE.

Proof. A node PULSEs only in Line 3.a, which is executed only when the value of $\mathcal{V}(\mathcal{BBB}_\Delta) = 1$. All nodes in $\mathcal{G}$ have not been subject to transient failures in the $\Delta_{node} = \Delta + 1$ beats preceding r. Therefore, $\mathcal{BBB}_\Delta$ has been initialized properly Δ beats ago, and during the Δ rounds of $\mathcal{BBB}_\Delta$'s execution, it has been executed properly by at least $n - f$ nodes. Hence, according to *Agreement* of $\mathcal{BBB}$, all nodes in $\mathcal{G}$ have the same value of $\mathcal{V}(\mathcal{BBB}_\Delta)$. Therefore, all nodes in $\mathcal{G}$ "act the same" when considering Line 3.a: either all of them execute Line 3.a or they all do not execute it. This holds for any beat after α (as long as $\mathcal{G}$ continues to contain $n-f$ correct nodes). Therefore, at any such beat $r \in [\alpha, \beta]$, either all nodes in $\mathcal{G}$ PULSE or they all do not PULSE. □

Lemma 2. *$\forall\beta \geq \alpha+\Delta+Cycle'$: If the system is* CORRECT(α, β)*, then at some beat $r \in [\alpha, \beta]$ all nodes in $\mathcal{G}$* PULSE.

Proof. According to the previous lemma, all nodes in $\mathcal{G}$ PULSE together during the interval $[\alpha, \beta]$. Hence, if one of them PULSEd in the interval $[\alpha, \alpha + Cycle']$, all of them PULSEd, proving the claim.

Otherwise, consider the case where no node in $\mathcal{G}$ has PULSEd in the interval $[\alpha, \alpha + Cycle']$. Hence, at beat $\alpha + Cycle'$, for all the nodes in $\mathcal{G}$, the *Counter* variable has decreased to 0 or is negative. This is because *Counter* is bounded from above by $Cycle'$ (which is a fixed parameter of the protocol and is identical at all nodes); and as long as it holds a positive value, it decreases by 1 during each beat of the interval $[\alpha, \alpha + Cycle']$ (since no node PULSEs in that interval, *Counter* never increases). Since the interval is at least $Cycle'$ beats long, the value of *Counter* is less than (or equal to) 0.

Therefore, at beat $\alpha + Cycle'$ there are $|\mathcal{G}| \geq n - f$ correct nodes with $WantToPulse = 1$. Therefore, according to *Validity* of $\mathcal{BBB}$, Δ beats afterwards $\mathcal{V}(\mathcal{BBB}_\Delta)$ will output 1, and all nodes in $\mathcal{G}$ will PULSE. Thus, in the interval $[\alpha, \alpha+\Delta+Cycle']$ all nodes in $\mathcal{G}$ PULSE. Therefore, the claim holds for any beat interval $[\alpha, \beta]$, where $\beta \geq \alpha + \Delta + Cycle']$. □

Remark 4. The above lemma proves *p*rogress. That is, starting from any state, eventually there will be a PULSE.

Consider a system that is CORRECT(α, β) (for $\beta \geq \alpha + \Delta + Cycle'$), from Lemma 1, starting from beat α all nodes in $\mathcal{G}$ PULSE together. From Lemma 2, by beat $\alpha + \Delta + Cycle'$ all nodes in $\mathcal{G}$ have PULSEd. Therefore, by that round they have all reset their *Counter* values at the same beat. Since $WantToPulse$ depends solely on the value of *Counter*, and since all nodes in $\mathcal{G}$ agree on the output value of the $\mathcal{BBB}$ protocols, all nodes in $\mathcal{G}$ perform exactly the same lines of code following each beat in the beat interval $[\alpha + \Delta + Cycle', \beta]$.

Lemma 3. *$\forall\beta \geq \alpha+3\cdot\Delta+2\cdot Cycle'$: If the system is* CORRECT(α, β)*, then the system is $[\Delta, \Delta + Cycle']$-*PULSING *in the beat interval $[\alpha + 3 \cdot \Delta + 2 \cdot Cycle', \beta]$.*

Proof. According to previous lemmas, all correct nodes PULSE at some beat γ, no later than beat $\alpha + \Delta + Cycle'$; and from then on they all PULSE together. At beat γ they all reset their counters and will have positive $Counter$ values for at least $Cycle'$ rounds. Since $Cycle' > \Delta$, in the following Δ beats, the value of $WantToPulse$ will be 0, and hence $\mathcal{BBB}_1$ is initialized during these beats with the value 0. Therefore, once these values will emerge from $\mathcal{BBB}$, there will be a period of $Cycle' > \Delta$ with no pulses. That "quiet" period will start at beat $\gamma + \Delta$. This quiet period might be longer than $Cycle'$, if there were other PULSEs during the beat interval $[\gamma, \gamma + \Delta]$. In any case, a quiet period will commence at beat $\gamma + \Delta$ and will be at least $Cycle'$ beats long, and no more than $Cycle' + \Delta$ beats long.

Now consider what happens after this quiet period. Eventually, the value of $WantToPulse$ will be set to 1 (after no more than $Cycle' + \Delta$ beats), and will stay so until the next PULSE. Mark the beat at which all nodes in $\mathcal{G}$ set $WantToPulse$ to 1 as γ'. Notice that because the quiet period is greater than Δ, then once its values start emerging of $\mathcal{BBB}_\Delta$ there will be a quiet period for at least Δ beats. Hence, once $WantToPulse$ is set to 1, it will stay that way for Δ beats, until 1 comes out of $\mathcal{BBB}_\Delta$. This will happen at beat $\gamma' + \Delta$. Once this happens, there are Δ 1's "on the way" in the coming $\mathcal{BBB}$s. Therefore, there will be a PULSE for Δ beats. Due to the first PULSE, $WantToPulse$ will be 0 for all the Δ PULSE beats. After the last PULSE beat, $WantToPulse$ will be 0 for an additional $Cycle'$ beats. Afterwards, $WantToPulse$ will turn to 1, and will stay so for Δ beats. Thus there is a pattern of $WantToPulse$ being 0 for $\Delta + Cycle'$ beats then being 1 for Δ beats, and so on. Therefore, the PULSEing pattern will satisfy the requirement.

Note that the PULSEing pattern starts on beat $\gamma' + \Delta$, and the pattern continues (at least) until beat β. Hence, the system is $[\Delta, \Delta + Cycle']$-PULSING in the beat interval $[\gamma' + \Delta, \beta]$. Because $\gamma' \leq \gamma + Cycle' + \Delta$ and since $\gamma \leq \alpha + \Delta + Cycle'$, we conclude that $\gamma' + \Delta \leq \alpha + 3 \cdot \Delta + 2 \cdot Cycle'$, as required. □

Remark 5. The above lemma shows that the convergence time of the PULSEing algorithm depends on the value of $Cycle$. However, since for clock synchronization the value of $Cycle$ is in the order of Δ, the convergence of the clock synchronization will depend on Δ and not on the value of *max-clock* (the wrap around value of the digital clock).

The following theorem states that we have constructed a $[\Delta, \Delta + Cycle']$-PULSER.

Theorem 1. *The Large-Cycle-Pulser algorithm is a* $[\Delta, \Delta + Cycle']$-PULSER.

Proof. By Lemma 3, once there are enough nodes that have not been subject to transient failures for $3 \cdot \Delta + 2 \cdot Cycle'$ beats, the system becomes $[\Delta, \Delta + Cycle']$-PULSING for the beat interval $[\gamma' + \Delta, \beta]$. This is true for any $\beta \geq \alpha + 3 \cdot \Delta + 2 \cdot Cycle'$. Hence, as long as the system is coherent, once the system is $[\Delta, \Delta + Cycle']$-PULSING in the beat interval $[\gamma' + \Delta, \beta]$, it is also $[\Delta, \Delta + Cycle']$-PULSING in the beat interval $[\gamma' + \Delta, \beta + 1]$; and therefore Large-Cycle-Pulser algorithm is a $[\Delta, \Delta + Cycle']$-PULSER. □

5 A $[Cycle]$-PULSER for $Cycle > 0$

In the previous section a $[\Delta, \Delta + Cycle']$-PULSER was presented, for any value of $Cycle' > \Delta$. Now a general way to transform a $[\phi, \psi]$-PULSER into a $[\phi + \psi]$-PULSER is given. Combining this with the previous result produces a $[2 \cdot \Delta + Cycle']$-PULSER. Since $Cycle' > \Delta$, this technique constructs a $[Cycle]$-PULSER, for any $Cycle > 3 \cdot \Delta$. In Subsection 5.2 this requirement is eliminated, and the objective of building $[Cycle]$-PULSER is achieved for any $Cycle > 0$.

5.1 $[\phi, \psi]$-PULSER to $[\phi + \psi]$-PULSER

Given a $[\phi, \psi]$-PULSER $\mathcal{A}$, the algorithm $\mathcal{B}$ in Figure 2 uses $\mathcal{A}$ as a black-box:

Algorithm $[\phi + \psi]$-PULSER ** executed repeatedly at each beat */*

1. execute a single round of $\mathcal{A}$;
2. if $\mathcal{A}$ PULSEd at the current beat and $\mathcal{A}$ did not PULSE at the previous beat, then $\mathcal{B}$ PULSEs at the current beat.

Fig. 2. An algorithm that transforms a $[\phi, \psi]$-PULSER into a $[\phi + \psi]$-PULSER.

Note that the above algorithm $\mathcal{B}$ does not rely on anything other than the output of $\mathcal{A}$ in the current and previous beats. Hence, if $\mathcal{A}$ is self-stabilizing, so is $\mathcal{B}$.

Theorem 2. *The algorithm $\mathcal{B}$ is a $[\phi + \psi]$-PULSER.*

Proof. $\mathcal{A}$ is a $[\phi, \psi]$-PULSER, hence, it PULSEs in a pattern of ϕ pulses, then ψ quiet rounds. Therefore, once every $\phi + \psi$ beats, there is a transition from not PULSEing to PULSEing. Thus, the PULSEing output of $\mathcal{A}$, implies that exactly once every $\phi + \psi$ beats it holds that $\mathcal{A}$ PULSEd at the current beat, and did not PULSE at the previous beat. This is continuously true (as long as $\mathcal{A}$ continues to PULSE), which implies that the proposed algorithm $\mathcal{B}$ will PULSE exactly once every $\phi + \psi$ beats, in a pattern of a single PULSE, and then $\phi + \psi - 1$ beats of quiet rounds.

Since $\mathcal{A}$ is a $[\phi, \psi]$-PULSER, starting from an arbitrary state, it eventually starts PULSEing in the required pattern, and continues so as long as the system is coherent. Hence, the above algorithm $\mathcal{B}$ will eventually start PULSEing in the expected pattern, and will continue so as long as the system is coherent. Hence it is a $[\phi + \psi]$-PULSER. □

5.2 Case $Cycle \leq 3 \cdot \Delta$

Building upon the $[2 \cdot \Delta + Cycle']$-PULSER, $\mathcal{B}$, from the previous subsection, a $[Cycle]$-PULSER, $\mathcal{C}$, for any $Cycle \leq 3 \cdot \Delta$ is presented in Figure 3.

Algorithm $[Cycle \leq 3 \cdot \Delta]$-PULSER /* *executed repeatedly at each beat* */

/* *set* $Cycle' > \Delta$ *to be such that* $Cycle' + 2 \cdot \Delta$ *is divisible by* $Cycle$ */

1. execute $\mathcal{B}$;
2. if $\mathcal{B}$ PULSEd at the current beat then
   $Counter := Cycle' + 2 \cdot \Delta$;
3. if $Counter$ is divisible by $Cycle$ then
   $\mathcal{C}$ PULSEs at the current beat;
4. $Counter := Counter - 1$.

Fig. 3. A $[Cycle]$-PULSER algorithm for $1 \leq Cycle \leq 3 \cdot \Delta$.

Theorem 3. *The algorithm* $\mathcal{C}$ *is a* $[Cycle]$-PULSER *for any* $1 \leq Cycle \leq 3 \cdot \Delta$.

Proof. Since $\mathcal{B}$ is a $[Cycle' + 2 \cdot \Delta]$-PULSER, starting from an arbitrary state, eventually it starts PULSEing in a pattern of a single PULSE, and then $Cycle' + 2 \cdot \Delta - 1$ beats of quiet rounds (and continues so as long as the system is coherent). Therefore, eventually, all correct nodes will see the same PULSEing output from $\mathcal{B}$. Hence, each time $\mathcal{B}$ pulses, all correct nodes set $Counter$ to $Cycle' + 2 \cdot \Delta$, and have the same value of $Counter$ at each beat (because they all set it together, and decrease it together). Thus, each time a correct node enters Line 3, all correct nodes do the same. Therefore, all correct nodes have $\mathcal{C}$ PULSE together. Lastly, since each $Cycle$ beats $Counter$ will be divisible by $Cycle$, $\mathcal{C}$ PULSEs once every $Cycle$ beats.

Therefore, for each PULSE of $\mathcal{B}$ we have $\frac{2 \cdot \Delta + Cycle'}{Cycle}$ PULSEs of $\mathcal{C}$. Due to the choice of $Cycle'$ such that $2 \cdot \Delta + Cycle'$ is divisible by $Cycle$, the PULSEs are nicely aligned with the PULSEs of $\mathcal{B}$; and therefore, the above algorithm $\mathcal{C}$ is a $[Cycle]$-PULSER. □

Theorem 4. *For any* $Cycle > 0$, *a* $[Cycle]$-PULSER *can be constructed.*

Proof. If $Cycle > 3 \cdot \Delta$, then set $Cycle' := Cycle - 2 \cdot \Delta$. By Theorem 1, construct a $[\Delta, \Delta + Cycle']$-PULSER and by Theorem 2 construct a $[2 \cdot \Delta + Cycle']$-PULSER. According to the choice of $Cycle'$ the required $[Cycle]$-PULSER is constructed.

If $Cycle \leq 3 \cdot \Delta$, calculate $Cycle'$ such that $Cycle' > \Delta$ and $\frac{2 \cdot \Delta + Cycle'}{Cycle}$ is an integer number. Now, by Theorem 1 build a $[\Delta, \Delta + Cycle']$-PULSER. From Theorem 2 construct a $[2 \cdot \Delta + Cycle']$-PULSER. Finally, from the above algorithm construct an algorithm that is a $[Cycle]$-PULSER, as required. □

6 Network Connectivity

The above discussion did not assume anything about the network connectivity. More precisely, the only connectivity assumption was about the behavior of the $\mathcal{BBB}$ protocol. That is, whatever connectivity $\mathcal{BBB}$ requires to operate properly, is the required connectivity in order for the $[Cycle]$-PULSER construction to work properly.

In [5] it is shown that *Byzantine* agreement is achievable if and only if:

1. f is less than one-third of the total number of nodes in the system.
2. f is less than one-half of the connectivity of the system (that is, between any two nodes there are at least $2 \cdot f + 1$ distinct paths).

These lower bounds clearly hold for *Byzantine* consensus. Therefore, since $\mathcal{BBB}$ is implemented by executing *Byzantine* Consensus for each node's input value, $\mathcal{BBB}$ can be tolerant to up to $\frac{n-1}{3}$ *Byzantine* faults. In addition $\mathcal{BBB}$ can work properly even if the connectivity graph is not fully connected, but rather there are at least $2 \cdot f + 1$ distinct paths between any two non-faulty nodes.

Remark 6. As noted in [5], the nodes are required to know the connectivity graph while executing the algorithm. This implies, due to self-stabilization, that each node has the network connectivity as incorruptible data.[5]

Since the PULSEing algorithm presented in this paper depends solely on $\mathcal{BBB}$ for communication with other nodes, it is tolerant up to $\frac{n-1}{3}$ *Byzantine* faults and can operate in a network where there are at least $2 \cdot f + 1$ distinct paths between any two nodes, and it is optimal with respect to these two parameters.

Previous synchronization algorithms do not easily extend to operate in a network that is not fully connected. This is a result of the dependency of their "current state" on messages received in the "current round"; in a network that is not fully connected, such messages are received $\mathcal{D}$ rounds later, where $\mathcal{D}$ is the diameter of the network.

For example, in [12], the *DigiClock* value depends on the values sent in the current round. Therefore, if the network is not fully connected, node p does not receive messages from node p' that is not his neighbor, in the same round. Hence, p cannot change its current state according to the algorithm's definition. This does not mean that previous algorithms cannot be transformed to operate in such a setting, just that it is not straightforward.

7 Complexity Analysis

Using PULSEing for clock synchronization leads using a *Cycle* that is in the order of Δ. Hence, the PULSEing algorithm presented in the previous sections converges in $O(\Delta)$ beats. If the system is fully connected, then $\Delta = 2f + 3$, because efficient implementations of *Byzantine* consensus require about $2f + 3$ rounds. Therefore, convergence is reached in $O(f)$ beats.

If the system is not fully connected, as discussed in the previous section, and the diameter is $\mathcal{D}$, then $\Delta = \mathcal{D} \cdot 2 \cdot (f + 1)$. Therefore, convergence is reached in $O(\mathcal{D} \cdot f)$ beats.

[5] One can somewhat relax this assumption, but then either a flooding algorithm needs to be used, or one needs to come up with an algorithm that finds enough independent paths on the fly - despite the *Byzantine* behavior; we are not aware of any self-stabilizing algorithm to do that.

Considering message complexity, at each beat Δ $\mathcal{BBB}$s are executed simultaneously. Since $\mathcal{BBB}$ can be implemented via *Byzantine* consensus (see Remark 2), it requires n^2 messages at each beat. Hence we have that the message complexity at each beat is $O(f \cdot n^2)$. Note that one can use early stopping agreements. Such agreements will use less messages, if the number of faults is small, but will still take the same worse case time.

8 The Digital Clock Synchronization Problem

In the digital clock synchronization problem, each node has a variable *DigiClock*, and the objective is to have all correct nodes agree on the value of *DigiClock* and increase it by one at each beat. A more detailed discussion of this problem (along with a solution) is given in [12].

The digital clock synchronization problem is equivalent to the PULSEing problem. Given an algorithm that solves the digital clock synchronization problem, simply PULSE every time the *DigiClock* variable is divisible by *Cycle*. This produces a [*Cycle*]-PULSER algorithm.

The other direction is a bit more complicated. Given a $[f + 2]$-PULSER algorithm, every PULSE execute a *Byzantine* agreement on what the *DigiClock* value will be in the next PULSE. In addition, each beat *DigiClock* is increased by 1, and when the *Byzantine* agreement terminates, the *DigiClock* is set to the agreement value (similar to [6]). This way, all nodes agree on the value of *DigiClock* and increase it by one at each beat.

Note that the digital clock synchronization problem has been solved directly in [12] for $f < \frac{n}{4}$ and assuming a fully connected graph. Due to the equivalence to the PULSEing problem, a digital clock synchronization algorithm can be built with an underlying PULSEing algorithm presented in this paper, which supports $f < \frac{n}{3}$ and assumes only that there are $2 \cdot f + 1$ distinct paths between any two nodes. That produces a digital clock synchronization algorithm that is optimal in these two aspects.

9 *Byzantine* Tolerant Stabilizer

We now present briefly how a stabilizer can be built using the PULSEing algorithm provided in the above sections. The stabilizer will *stabilize* a *Byzantine* tolerant algorithm $\mathcal{A}_0$. That is, given a *Byzantine* tolerant algorithm $\mathcal{A}_0$ that is not self-stabilizing, the stabilizer will transform it into a self stabilizing version of $\mathcal{A}_0$ (preserving the *Byzantine* tolerance).

Clearly, not all algorithms can be viewed as self-stabilizing. E.g. an algorithm that is allowed to do some action ACT only once, cannot be a self-stabilizing algorithm. We do not discuss here the requirements of an algorithm $\mathcal{A}_0$ so that it can be stabilized. For a more in depth discussion of such requirements, refer to [10] and [13]. In the following, it is assumed that the *Byzantine* tolerant algorithm $\mathcal{A}_0$ has a meaning as a self-stabilizing algorithm.

Intuitively, every so often, all nodes will collect a global snapshot $\mathcal{S}$ of the local states of all nodes. Then, all nodes inspect $\mathcal{S}$ for any inconsistencies. If any are found, all nodes reset their local state to some consistent state.

Given a general *Byzantine* tolerant algorithm $\mathcal{A}_0$, we construct an algorithm Byz-State-Check. Byz-State-Check gathers a global snapshot of the local states at each node and ensures that the local states are consistent. In addition if the states were consistent to start with, then Byz-State-Check does not alter them. That is, Byz-State-Check alters the local states to a consistent state, only if required. Figure 4 presents the algorithm Byz-State-Check.

Algorithm **Byz-State-Check** */* executed at node p*/*

1. execute a *Byzantine* agreement on local state of $\mathcal{A}_0$;
2. Δ_{agree} beats after the the beginning of the execution of line 1:
   (a) if $\mathcal{S}$ represents a legal state
   repair local state if it is inconsistent with $\mathcal{S}$;
   (b) otherwise
   reset local state.

Fig. 4. A *Byzantine* tolerant state validation and reset.

Remark 7. Δ_{agree} is an upper bound on the number of rounds it takes to execute *Byzantine* agreement ($2f + 3$ for a typical efficient implementation). Since all correct nodes wait Δ_{agree} beats from entering line 1 until entering line 2, it is ensured that all correct nodes enter line 2 after they see the same global snapshot $\mathcal{S}$.

Given a general *Byzantine* tolerant algorithm $\mathcal{A}_0$, a $[\Delta_{agree}+1]$-PULSER $\mathcal{P}$ and a Byz-State-Check algorithm $\mathcal{C}$, the algorithm SS-Byz-Stabilizer is constructed, as in Figure 5.

Algorithm **SS-Byz-Stabilizer** */* executed at each beat */*
/ $\mathcal{A}_0$ is the algorithm to be stabilized */*
/ $\mathcal{C}$ is an instance of Byz-State-Check*/*
/ $\mathcal{P}$ is a $[\Delta_{agree} + 1]$-PULSER */*

1. execute a single round of $\mathcal{A}_0$;
2. execute a single round of $\mathcal{C}$;
3. execute a single beat of $\mathcal{P}$;
4. if $\mathcal{P}$ PULSEd this beat re-initialize $\mathcal{C}$.

Fig. 5. A Self-stabilizing *Byzantine* tolerant Stabilizer.

Theorem 5. *SS-Byz-Stabilizer transforms a* Byzantine *tolerant algorithm* $\mathcal{A}_0$ *into Self-stabilizing* Byzantine *tolerant algorithm.*

Proof. $\mathcal{P}$ is a $[\Delta_{agree}+1]$-PULSER. Hence, eventually it starts PULSEing $\Delta_{agree}+1$ beats apart. When this happens, $\mathcal{C}$ is re-executed periodically, and terminates between such 2 executions . Hence, $\mathcal{C}$ performs correctly. This means that the local states of $\mathcal{A}_0$ will be consistent. And we have that starting from any initial state of $\mathcal{A}_0$'s local states, eventually $\mathcal{A}_0$'s local states are consistent. □

10 Acknowledgments

We would like to thank Ariel Daliot for helpful discussions and insightful comments.

References

1. Y. Afek and S. Dolev. Local stabilizer. In *Proc. of the 5th Israeli Symposium on Theory of Computing Systems (ISTCS97)*, Bar-Ilan, Israel, Jun 1997.
2. A. Arora, S. Dolev, , and M.G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.
3. A. Daliot and D. Dolev. Self-stabilization of byzantine protocols. In *In Proc. of the 7th Symposium on Self-Stabilizing Systems (SSS'05)*, Barcelona, Spain, Oct 2005.
4. A. Daliot, D. Dolev, and H. Parnas. Linear time byzantine self-stabilizing clock synchronization. In *Proc. of 7th Int. Conference on Principles of Distributed Systems (OPODIS'03)*, La Martinique, France, Dec 2003. A corrected version appears in http://arxiv.org/abs/cs.DC/0608096.
5. D. Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3:14–30, 1982.
6. D. Dolev, J. Y. Halpern, B. Simons, and R. Strong. Dynamic fault-tolerant clock synchronization. *J. Assoc. Computing Machinery*, 42(1):143–185, Jan 1995.
7. S. Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Journal of Real-Time Systems*, 12(1):95–107, 1997.
8. S. Dolev and J. L. Welch. Wait-free clock synchronization. *Algorithmica*, 18(4):486–511, 1997.
9. S. Dolev and J. L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. *Journal of the ACM*, 51(5):780–799, 2004.
10. A. S. Gopal and K. J. Perry. Unifying self-stabilization and fault-tolerance. In *IEEE Proceedings of the 12th annual ACM symposium on Principles of distributed computing*, Ithaca, New York, 1993.
11. T. Herman. Phase clocks for transient fault repair. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1048–1057, 2000.
12. E. N. Hoch, D. Dolev, and A. Daliot. Self-stabilizing byzantine digital clock synchronization. In *Proc. of 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'06)*, Dallas, Texas, Nov 2006.
13. S. Katz and K. J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.
14. M. Nesterenko and A. Arora. Dining philosophers that tolerate malicious crashes. In *22nd Int. Conference on Distributed Computing Systems*, 2002.
15. M. Nesterenko and A. Arora. Tolerance to unbounded byzantine faults. In *SRDS*, pages 22–, 2002.
16. M. Papatriantafilou and P. Tsigas. On self-stabilizing wait-free clock synchronization. *Parallel Processing Letters*, 7(3):321–328, 1997.
17. Y. Sakurai, F. Ooshita, and T. Masuzawa. A self-stabilizing link-coloring protocol resilient to byzantine faults in tree networks. In *OPODIS*, pages 283–298, 2004.