Self-stabilizing Byzantine Digital Clock Synchronization

Ezra N. Hoch, Danny Dolev*, and Ariel Daliot

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

Abstract. We present a scheme that achieves self-stabilizing *Byzantine* digital clock synchronization assuming a "synchronous" system. This synchronicity is established by the assumption of a common "beat" delivered with a regularity in the order of the network message delay, thus enabling the nodes to execute in lock-step. The system can be subjected to severe transient failures with a permanent presence of *Byzantine* nodes. Our algorithm guarantees eventually synchronized digital clock counters, i.e. common increasing integer counters associated with each beat. We then show how to achieve regular clock synchronization, progressing at real-time rate and with high granularity, from the synchronized digital clock counters.

There is one previous self-stabilizing *Byzantine* clock synchronization algorithm, which also converges in linear time (relying on an underlying pulse mechanism), but it requires to execute and terminate *Byzantine* agreement in between consecutive pulses. Such a scheme, although it does not assume a synchronous system, cannot be easily transformed to a synchronous system in which the pulses (beats) are in the order of the message delay time apart. The only other digital clock synchronization algorithm operating in a similar synchronous model converges in expected exponential time. Our algorithm converges (deterministically) in linear time.

1 Introduction

Clock synchronization is a very fundamental task in distributed systems. The vast majority of distributed tasks require some sort of synchronization; and clock synchronization is a very straightforward and intuitive tool for supplying this. It thus makes sense to require an underlying clock synchronization mechanism to be highly fault-tolerant. A self-stabilizing algorithm seeks to attain synchronization once lost; a *Byzantine* algorithm assumes synchronization is never lost and focuses on containing the influence of the permanent presence of faulty nodes.

We consider a system in which the nodes execute in lock-step by regularly receiving a common "pulse" or "tick" or "beat". We will use the "beat" notation

* Part of the work was done while the author visited Cornell University. This research was supported in part by ISF, NSF, CCR, and AFSOR.

A.K. Datta and M. Gradinariu (Eds.): SSS 2006, LNCS 4280, pp. 350–362, 2006.

in order to stay clear of any confusion with "pulse synchronization" or "clock ticks". Should the beat interval be at least as long as the worst-case execution-time for terminating *Byzantine* agreement, then the system becomes, in a sense, similar to classic non-stabilizing systems, in which algorithms are initialized synchronously.[1] On the other hand, should the pulse interval length be in the order of the communication end-to-end delay, then the problem becomes agreeing on beat-counters or on "special" beats among the frequent common beats received.

The digital clock synchronization problem is to ensure that eventually all the correct nodes hold the same value of the beat counter (*digital clock*) and as long as enough nodes remain correct, they will continue to hold the same value and to increase it by one following each beat.

The mode of operation of the scheme proposed in this paper is to initialize at every beat *Byzantine* consensus on the digital clocks. Thus, after a number of beats (or *rounds*), which equals the bound for terminating *Byzantine* consensus, say Δ, all correct nodes have identical views on an agreed digital clock value of Δ rounds ago. Based on this global state, a decision is taken at every node how to adjust its local digital clock. At each beat, if the two most recently terminated consensus instances show consecutive digital clock values then the node increments its digital clock and initializes a new consensus instance on this updated digital clock value. If the digital clocks are not synchronized, the new consensus instance is initialized with the "zero" value. The algorithm converges within $3 \cdot \Delta$ rounds. We use "clock"and "digital clock" interchangeably.

Related work: We present a self-stabilizing *Byzantine* clock synchronization algorithm that assumes that common beats are received synchronously (simultaneously) and in the order of the message delay apart. The clocks progress at real-time rate. Thus, when the clocks are synchronized, in-spite of permanent *Byzantine* faults, the clocks may accurately estimate real-time.[2] Following transient failures, and with on-going *Byzantine* faults, the clocks will synchronize within a finite time and will progress at real-time rate, although the actual clock-reading values will not be directly correlated to real-time. Many applications utilizing the synchronization of clocks do not really require the exact real-time notion (see [12]). In such applications, agreeing on a common clock reading is sufficient as long as the clocks progress within a linear envelope of any real-time interval. Clock synchronization in a similar model has earlier been denoted as "digital clock synchronization" ([1,8,10,14]) or "synchronization of phase-clocks" ([11]), in which the goal is to agree on continuously incrementing counters associated with the beats. The convergence time in those papers is not linear, whereas in our solution it is linear.

The additional requirement of tolerating permanent *Byzantine* faults poses a special challenge for designing self-stabilizing distributed algorithms due to the capability of malicious nodes to hamper stabilization. This difficulty may be

[1] See [6] for such a self-stabilizing *Byzantine* clock synchronization algorithm, which executes on top of a self-stabilizing *Byzantine* pulse-synchronization primitive.

[2] All the arguments apply also to the case where there is a small bounded drift among correct clocks.

indicated by the remarkably few algorithms resilient to both fault models (see [3] for a short review). The digital clock synchronization algorithms in [9] are, to the best of our knowledge, the first self-stabilizing algorithms that are tolerant to *Byzantine* faults. The randomized algorithm, presented in [9], operating in the same model as in the current paper, converges in expected exponential time.

In [6] we have previously presented a self-stabilizing *Byzantine* clock synchronization algorithm, which converges in linear time and does not assume a synchronous system. That algorithm executes on top of a pulse synchronization primitive with intervals that allow to execute *Byzantine* agreement in between. The solution presented in the current paper only assumes that the (synchronously received) beats are on the order of the message delay apart and also converges in linear time. In [2] and [5] two pulse synchronization procedures are presented that do not assume any sort of prior synchronization such as common beats. One is biologically inspired and the other utilizes a self-stabilizing Byzantine agreement algorithm developed in [4]. Both these pulse synchronization algorithms are complicated and have complicated proofs, while the current solution is achieved in a relatively straightforward manner and its proofs are simpler. Due to the relative simplicity of the algorithm, formal verification methods, as were used in [13], can be used to increase the confidence in the correctness of the proposed algorithm. An additional advantage of the current solution is that it can be implemented without the use of local physical timers at the nodes.

2 Model

We consider a fully connected network of n nodes. 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 eventually leave the system in an arbitrary state. The algorithm tolerates a permanent fraction, $f < \frac{n}{4}$, of faulty *Byzantine* nodes.

We say that a node is *Byzantine* if it does not follow the instructed algorithm and *non-Byzantine* otherwise. Thus, a node that has crashed or experiences some other fault that does not allow it to exactly follow the algorithm as instructed, is considered *Byzantine*, even if it does not behave maliciously. A non-*Byzantine* node will therefore be called *non-faulty*.

We assume that the network has bounded time on message delivery when it behaves coherently. Nodes are instructed to send their messages immediately after the delivery of a beat from the global beat system. We assume that message delivery and the processing involved can be completed between two consecutive global beats. 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.

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 the nodes can be arbitrarily large and the communication network may behave

arbitrarily. Eventually the system behaves coherently again. At such a state a non-faulty node may find itself in an arbitrary state.

Definition 1. *The system is* coherent *if there are at most f Byzantine nodes, messages arrive and are processed at their non-faulty destinations between two consecutive beats.*

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.*[3]

Denote by $DigiClock_p(r)$ the value of the digital clock at node p at beat r. We say that the system is in a *synchronized_state* if for all correct nodes the value of their *DigiClock* is identical.

Definition 3. The digital-clock synchronization problem

Convergence: *Starting from an arbitrary system state, the system reaches a synchronized_state after a finite time.*

Closure: *If at beat r the system is in a synchronized_state then for every r', $r' \geq r$,*

1. *the system is in a synchronized_state at beat r'; and*
2. *$DigiClock(r') = (DigiClock(r)+r'-r) \ mod \ overlap$,*[4] *at each correct node.*

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

2.1 The Byzantine Consensus Protocol

Our digital clock synchronization algorithm utilizes a *Byzantine* consensus protocol as a sub-routine. We will denote this protocol by $\mathcal{BC}$. We require the regular conditions of Consensus from $\mathcal{BC}$, in addition to one additional requirement. That is, in $\mathcal{BC}$ the following holds:

1. Agreement: All non-faulty nodes terminate $\mathcal{BC}$ with the same output value.
2. Validity: If all non-faulty nodes have the same initial value v, then the output value of all non-faulty nodes is v.
3. Termination: All non-faulty nodes terminate $\mathcal{BC}$ within Δ rounds.
4. Solidarity. If the non-faulty nodes agree on a value v, such that $v \neq \perp$ (where $\perp$ denotes a non-value), then there are at least $n - 2 \cdot f$ non-faulty nodes with initial value v.

[3] The assumed value of Δ_{node} in the current paper will be defined later.

[4] "overlap" is the wrap around of the variable *DigiClock*. All additions to *DigiClock* in the rest of the paper are assumed to be (*mod overlap*).

Remark 1. Note that for $n > 4f$ the "solidarity" requirement implies that if the *Byzantine* consensus is started with at most $\frac{n}{2}$ non-faulty nodes with the same value, then all non-faulty nodes terminate with the value $\perp$.

As we commented above, since $\mathcal{BC}$ requires the nodes to maintain a consistent state throughout the protocol, a non-faulty node that has recently recovered from a transient fault cannot 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.

In Appendix A we discuss how typical synchronous *Byzantine* consensus protocols can be used as such a $\mathcal{BC}$ protocol. The specific examples we discuss have two early stopping features: First, termination is achieved within $2f + 4$ of our rounds. If the number of actual *Byzantine* nodes is $f' \leq f$ then termination is within $2f' + 6$ rounds. Second, if all non-faulty nodes have the same initial value, then termination is within 4 rounds.

The symbol Δ denotes the bound on the number of rounds it takes $\mathcal{BC}$ to terminate at all correct nodes. That is, if $\mathcal{BC}$ has some early stopping feature, we still wait until Δ rounds pass. This means that the early stopping may improve the message complexity, but not the time complexity. By using the protocols in Appendix A, we can set $\Delta := 2f + 4$ rounds.

3 Digital Clock Synchronization Algorithm

The following digital clock synchronization algorithm tolerates up to $f < \frac{n}{4}$ concurrent *Byzantine* faults. We target for the digital clocks to be incremented by "1" every beat and we target at achieving synchronization of these digital clocks.

3.1 Intuition for the Algorithm

The idea behind our algorithm is that each node runs many simultaneous *Byzantine* consensus protocols. In each round of the algorithm it executes a single round in each of the *Byzantine* consensus protocols, but each *Byzantine* consensus protocol instance is executed with a different round number. That is, if $\mathcal{BC}$ takes Δ rounds to terminate, then the node runs Δ concurrent instances of it, where, for the first one it executes the first round, for the second it executes the second round, and in general for the i^{th} $\mathcal{BC}$ protocol it executes the i^{th} round. We index a $\mathcal{BC}$ protocol by the number of rounds passed from its invocation. When the Δ^{th} $\mathcal{BC}$ protocol is completed, a new instance of $\mathcal{BC}$ protocol is initiated. This mechanism, of executing concurrently Δ $\mathcal{BC}$ protocols, allows the non-faulty nodes to agree on the clock values as of Δ rounds ago. The nodes use the consistency of these values as of Δ rounds ago and the exchange of their current values to "tune" the future clock values.

3.2 Preliminaries

Given a *Byzantine* consensus protocol $\mathcal{BC}$, each node maintains the following variables and data structures:

Algorithm **Digital-SSByz-ClockSync** */* executed at each beat */*

1. for each $i \in \{1, .., \Delta\}$ do
   execute the i^{th} round of the $Agree[i]$ BC protocol;
2. send value of $DigiClock$ to all nodes and store the received clocks of other nodes in $ClockVec$;
3. set the following:
   (a) v:= the agreed value of $Agree[\Delta]$;
   (b) $DigiClock_{most}$:= the value appearing at least $\lfloor \frac{n}{2} \rfloor + 1$ times in $ClockVec$, and 0 otherwise;
4. (a) if $(v = 0)$ or $(v = v_{prev} + 1)$ then
   $DigiClock := DigiClock_{most} + 1$ $(mod\ overlap)$;
   (b) else
   $DigiClock := 0$;
5. for each $i \in \{2, ..., \Delta\}$ do
   $Agree[i] := Agree[i-1]$;
6. initialize $Agree[1]$ by invoking $\mathcal{BC}(DigiClock)$.
7. $v_{prev} := v$.

Fig. 1. The digital clock synchronization algorithm

1. $DigiClock$ holds the beat counter value at the node.
2. $ClockVec$ holds a vector containing the value of $DigiClock$ each node sent in the current round.
3. $DigiClock_{most}$ holds the value that appears at least $\frac{n}{2}+1$ times in $ClockVec$, if one exists.
4. $Agree[i]$ is the memory space of the i^{th} instance of $\mathcal{BC}$ protocol (the one initialized i rounds ago).
5. v holds the agreed value of the currently terminating $\mathcal{BC}$.
6. v_{prev} holds the value of v one round ago.

Note that all the variables are reset or recomputed periodically, so even if a node begins with arbitrary values in its variables, it will acquire consistent values. The consistency of the variable values used for $\mathcal{BC}$ are taken care of within that protocol.

Figure 1 presents the digital clock synchronization algorithm.

Remark 2. The model allows for only one message to be sent from node p to p' within one round (between two consecutive beats). The digital clock synchronization algorithm in Figure 1 requires sending two sets of messages in each round. Observe that the set of messages sent in Step 2 is not dependent on the operations taking place in Step 1, therefore, all messages sent by the algorithm during each round can be sent right after the beat and will arrive and processed before the next beat, meeting the model's assumptions.

Note that a "simpler" solution, such as running consensus on the previous $DigiClock$, adding to it $\Delta + 1$ and setting it as the current $DigiClock$ would not work, because for some specific initial values of $DigiClock$ the *Byzantine* nodes can cause the non-faulty nodes to get "stuck" in an infinite loop of alternating values.

4 Lemmata and Proofs

All the lemmata, theorems, corollaries and definitions hold only as long as the system is coherent. We assume that all nodes may start in an arbitrary state, and that from some time on, no more than f of them are *Byzantine*. We will denote by $\mathcal{G}$ a group of nodes that behave according to the algorithm, and that are not subject to (for some pre-specified number of rounds) any new transient faults. If, $|\mathcal{G}| \geq n - f$ and remain non-faulty for a long enough period of time ($\Omega(\Delta)$ global beats), then the system will converge.

For simplifying the notations, the proof refers to some "external" round number. The nodes do not maintain it, it is only used for the proofs.

Definition 4. *We say that the system is* $\textsc{Calm}(\alpha, \sigma)$, $\sigma > \alpha$, *if there is a set* $\mathcal{G}$, $|\mathcal{G}| = n - f$, *of nodes that are non-faulty during all rounds in the interval* $[\alpha, \sigma - 1]$.

The notation $\textsc{Calm}(\alpha, \sigma \geq \beta)$ denotes that $\textsc{Calm}(\alpha, \sigma)$ and $\sigma \geq \beta$. Specifically, the notation implies that the system was calm for at least β rounds. Notice that all nodes in $\mathcal{G}$ are considered correct when the system is $\textsc{Calm}(\alpha, \sigma \geq \Delta)$.

Note that in typical self-stabilizing algorithms it is assumed that eventually all nodes behave correctly, and therefore there is no need to define $\textsc{Calm}()$. In our context, since some nodes may never behave correctly, and additionally some nodes may recover and some may fail we need a sufficiently large subset of the nodes to behave correctly for sufficiently long time in order for the system to converge.

In the following lemmata, $\mathcal{G}$ refers to the set implied by $\textsc{Calm}(\alpha, \sigma)$, without stating so specifically.

Lemma 1. *If the system is* $\textsc{Calm}(\alpha, \sigma \geq \Delta + 1)$, *then for any round* β, $\beta \in [\alpha + \Delta + 1, \sigma]$, *all nodes in* $\mathcal{G}$ *have identical* v *values after executing Step 2 of Digital-SSByz-ClockSync.*

Proof. Irrespective of the initial states of the nodes in $\mathcal{G}$ at the beginning of round α (which is after the last transient fault in $\mathcal{G}$ occurred), the beats received from the global beat system will cause all nodes in $\mathcal{G}$ to perform the steps in synchrony. By the end of round α, all nodes in $\mathcal{G}$ reset $\mathcal{BC}$ protocol *Agree*[1].

Note that at each round another $\mathcal{BC}$ protocol will be initialized and after Δ rounds from its initialization each such protocol returns the same value at all nodes in $\mathcal{G}$, since all of them are non-faulty and follow the protocol. Hence, After $\Delta + 1$ rounds, the values all nodes in $\mathcal{G}$ receive as outputs of $\mathcal{BC}$ protocols are identical. Therefore v is identical at all $g \in \mathcal{G}$, after executing Step 2 of that round.

Since this claim depends only on the last $\Delta + 1$ rounds being "calm", the claim will continue to hold as long as no node in $\mathcal{G}$ experiences a transient fault. Thus, this holds for any round β, $\alpha + \Delta \leq \beta \leq \sigma$. □

Lemma 2. *If the system is* $\textsc{Calm}(\alpha, \sigma \geq \Delta + 2)$, *then for any round* $\beta \in [\alpha + \Delta + 1, \sigma]$, *either all nodes in* $\mathcal{G}$ *perform Step 4.a, or all of them perform Step 4.b.*

Proof. By Lemma 1, after the completion of Step 2 of round $\alpha+\Delta+1$ the value of v is the same at all nodes of $\mathcal{G}$, hence after an additional round the value of v_{prev} is the same at all nodes of $\mathcal{G}$. Since the decision whether to perform Step 4.a or Step 4.b depends only on the values of v, and v_{prev}, all nodes in $\mathcal{G}$ perform the same line (either 4.a or 4.b). Moreover, because this claim depends on the last $\Delta+2$ rounds being "calm", the claim will continue to hold as long as no node in $\mathcal{G}$ is subject to a fault. □

Denote $\Delta_1 := \Delta + 2$. All the following lemmata will assume the system is $\textsc{Calm}(\alpha, \beta)$, for rounds $\beta \geq \Delta_1$. Therefore, in all the following lemmata, we will assume that in each round β, all nodes in $\mathcal{G}$ perform the same Step 4.x (according to Lemma 2).

Lemma 3. *If the system is* $\textsc{Calm}(\alpha, \sigma \geq \Delta_1)$*, and if at the end of some* $\beta \geq \alpha + \Delta_1 - 1$*, all nodes in* $\mathcal{G}$ *have the same value of DigiClock, then at the end of any* β'*,* $\beta \leq \beta' \leq \sigma$*, they will have the same value of DigiClock.*

Proof. Since we consider only $\beta \geq \alpha + \Delta_1 - 1$, by Lemma 2 all nodes in $\mathcal{G}$ perform the same Step 4.x. For round $\beta' = \beta + 1$, the value of *DigiClock* can be changed at Lines 4.a or 4.b. If it was changed at 4.b then all nodes in $\mathcal{G}$ have the value 0 for *DigiClock*. If it was changed by Step 4.a, then because we assume that at round β all nodes in $\mathcal{G}$ have the same *DigiClock* value, and because $|\mathcal{G}| = n - f \geq \lfloor \frac{n}{2} + 1 \rfloor$, the value of $DigiClock_{most}$ computed at round β' is the same for all nodes in $\mathcal{G}$, and therefore, executing Step 4.a will produce the same value for *DigiClock* in round β' for all nodes in $\mathcal{G}$.

By induction, for any $\beta \leq \beta' \leq \sigma$, all nodes in $\mathcal{G}$ continue to agree on the value of *DigiClock*. □

Denote $\Delta_2 := \Delta_1 + \Delta + 1$. All the following lemmata will assume the system is $\textsc{Calm}(\alpha, \sigma)$, for $\sigma \geq \Delta_2$.

Lemma 4. *If the system is* $\textsc{Calm}(\alpha, \sigma \geq \Delta_2)$*, then at the end of any round* $\beta, \beta \in [\alpha + \Delta_2 - 1, \sigma]$*, the value of DigiClock at all nodes in* $\mathcal{G}$ *is the same.*

Proof. Consider any round $\beta' \in [\alpha + \Delta_1 - 1, \alpha + \Delta_1 + \Delta - 1]$. If at the end of β' all nodes in $\mathcal{G}$ hold the same *DigiClock* value, then from Lemma 3 this condition holds for any β, $\beta \in [\alpha + \Delta_2 - 1, \sigma]$. Hence, we are left to consider the case where at the end of any such β' not all the nodes in $\mathcal{G}$ hold the same value of *DigiClock*. This implies that Step 4.b was not executed in any such round β'. Also, if Step 4.a was executed during any such round β', and there was some *DigiClock* value that was the same at more than $\frac{n}{2}$ nodes in $\mathcal{G}$, then after the execution of Step 4.a, all nodes would have had the same *DigiClock* value. Hence, we assume that for all β', only Step 4.a was executed, and that no more than $\frac{n}{2}$ from $\mathcal{G}$ had the same *DigiClock* value.

Consider round $\beta'' = \alpha + \Delta_1 + \Delta$. The above argument implies that at round $\beta'' - \Delta$, Step 4.a was executed, and there were no more than $\frac{n}{2}$ nodes in $\mathcal{G}$ with the same *DigiClock* value. Since $\frac{n}{2} < n - 2 \cdot f$, the "solidarity" requirement of

$\mathcal{BC}$ implies that the value entered into v at round β'' is $\perp$. Hence, at round β'' Step 4.b would be executed.

Therefore, during one of the rounds $\beta \in [\alpha + \Delta_1 - 1, \alpha + \Delta_1 + \Delta]$, all the nodes in $\mathcal{G}$ have the same value of *DigiClock*, and from Lemma 3 this condition holds for all rounds, until σ. □

Remark 3. The requirement that $f < \frac{n}{4}$ stems from the proof above. That is because we require that $\frac{n}{2} < n - 2 \cdot f$ (to be able to use the "solidarity" requirement of $\mathcal{BC}$). We note that this is the only place that the requirement $f < \frac{n}{4}$ appears, and that it is a question for future research whether this can be improved to the known lower bound of $f < \frac{n}{3}$.

Corollary 1. *If the system is* $\textsc{Calm}(\alpha, \sigma \geq \Delta_2)$*, then for every round* β, $\beta \in [\alpha + \Delta_2 - 1, \sigma - 1]$, *one of the following conditions holds:*

1. *The value of DigiClock at the end of round* $\beta + 1$ *is "0" at all nodes in* $\mathcal{G}$.
2. *The value of DigiClock at the end of round* $\beta + 1$ *is identical at all nodes in* $\mathcal{G}$ *and it is the value of DigiClock at the end of round* β *plus "1".*

Lemma 5. *If the system is* $\textsc{Calm}(\alpha, \sigma \geq \Delta_2 + \Delta)$*, then for every round* $\beta \in [\alpha + \Delta_2 + \Delta - 1, \sigma]$, *Step 4.b is not executed.*

Proof. By Corollary 1, for all rounds β, $\beta \in [\alpha + \Delta_2 - 1, \sigma - 1]$ one of the two conditions of the *DigiClock* values holds. Due to the "validity" property of $\mathcal{BC}$, after Δ rounds, the value entered into v is the same *DigiClock* value that was at the nodes in $\mathcal{G}$, Δ rounds ago. Therefore, after Δ rounds, the above conditions hold on the value of v, v_{prev}. Hence, for any round β, $\beta \in [\alpha + \Delta_2 + \Delta - 1, \sigma]$ one of the conditions holds on v, v_{prev}. Since for both of these conditions, Step 4.a is executed, Step 4.b is never executed for such a round β. □

Corollary 2. *If the system is* $\textsc{Calm}(\alpha, \sigma \geq \Delta_2 + \Delta)$*, then for every round* β, $\beta \in [\alpha + \Delta_2 + \Delta - 1, \sigma]$, *it holds that all nodes in* $\mathcal{G}$ *agree on the value of DigiClock and increase it by "1" at the end of each round.*

Corollary 2 implies, in a sense, the convergence and closure properties of algorithm Digital-SSByz-ClockSync.

Theorem 1. *From an arbitrary state, once the system stays coherent and there are* $n - f$ *correct nodes that are non-faulty for* $3\Delta + 3$ *rounds, the Digital-SSByz-ClockSyncensures converges to a synchronized_ state. Moreover, as long as there are at least* $n - f$ *correct nodes at each round the closure property also holds.*

Proof. The conditions of the theorem implies that the system satisfies $\textsc{Calm}(\alpha, \sigma \geq \Delta_2 + \Delta)$. Consider the system at the end of round $\Delta_2 + \Delta$ and denote by $\bar{\mathcal{G}}$ a set of $n - f$ correct nodes implied by $\textsc{Calm}(\alpha, \sigma \geq \Delta_2 + \Delta)$. Consider all the Δ instances of $\mathcal{BC}$ in their memory. Denote by $\mathcal{BC}_i$ the instance of $\mathcal{BC}$ initialized i $(0 \leq i \leq \Delta - 1)$ rounds ago. By Lemma 5, Step 4.b is not going to be executed (if the nodes in $\bar{\mathcal{G}}$ will continue to be non-faulty). Therefore, at the end of the current round,

1. the set of inputs to each $\mathcal{BC}_i$ contained at least $\lfloor \frac{n}{2} \rfloor + 1$ identical values from non-faulty nodes, when it was initialized (denote that value I_i);
2. for every i, $0 \leq i \leq \Delta - 1$, either $I_i = I_{i+1}$ or $I_i = 0$;
3. I_0 is the value that at least $\lfloor \frac{n}{2} \rfloor + 1$ non-faulty hold in their DigiClockat the end of the current round.

The first property holds because otherwise, by the "solidarity" property of $\mathcal{BC}$, the agreement in that $\mathcal{BC}$ will be on $\perp$ and Step 4.b will be executed. The second property holds because otherwise Step 4.b will be executed. The third property holds since this is the value they initialized the last $\mathcal{BC}$ with.

Observe, that each $\mathcal{BC}_i$ will terminate in $\Delta - i$ rounds with a consensus agreement on I_i, as long as there are $n - f$ non-faulty nodes that were non-faulty throughout its Δ rounds of execution. Thus, under such a condition, for that to happen some nodes from $\bar{\mathcal{G}}$ may fail and still the agreement will be reached. Therefore, Corollary 2 holds for each node that becomes correct, i.e., was non-faulty for Δ rounds, because it will compute the same values as all the already correct nodes.

By a simple induction we can prove that the three properties above will hold in any future round, as long as for each $\mathcal{BC}$ there are $n - f$ non-faulty nodes that executed it.

Thus, the three properties imply that the basic claim in Corollary 2 will continue to hold, which completes the proof of the convergence and closure properties of the system. □

Note that if the system is stable and the actual number of *Byzantine* faults f' is less than f, then Theorem 1 implies that any non-faulty node that is not in $\mathcal{G}$ (there are no more than $f - f'$ such nodes) synchronizes with the *DigiClock* value of nodes in $\mathcal{G}$ after at most Δ global beats from its last transient fault.

5 Complexity Analysis

The clock synchronization algorithm presented above converges in $3 \cdot \Delta + 3$ rounds. That is, it converges in $\Omega(f)$ rounds (since $\Delta = 2 \cdot f + 4$ for our $\mathcal{BC}$ of choice).

Once the system converges, and there are at least $|\mathcal{G}| = n - f$ correct nodes, $\mathcal{BC}$ protocol will stop executing after 4 rounds for all nodes in $\mathcal{G}$ (due to the early stopping feature of $\mathcal{BC}$ we use). During each round of $\mathcal{BC}$, there are n^2 messages exchanged. Note that we execute Δ concurrent $\mathcal{BC}$ protocols; hence, over a period of Δ rounds, $\Delta \cdot 4 \cdot n^2$ messages. Therefore, the amortized message complexity per round is $O(n^2)$. Note that the early stopping of $\mathcal{BC}$ does **not** improve the convergence rate. It only improves the amortized message complexity.

6 Discussion

A Scheme for "Rotating Consensuses". Although the current work is presented as a digital clock synchronization algorithm, it actually surfaces a more

general scheme for "rotating" *Byzantine* consensus instances, which allows all non-faulty nodes to have a global "snapshot" of the state that was several rounds ago. This mechanism is self-stabilizing and tolerates the permanent presence of *Byzantine* nodes. This mechanism ensures that all non-faulty nodes decide on their next step at the next round, based on the same information.

Our usage of consensus provides agreement on a global "snapshot" of some global state. By replacing each *Byzantine* consensus with n *Byzantine* agreements, this mechanism can provide a global "snapshot" of the states of all the nodes several rounds ago. That is, instead of agreeing on a single state for the entire system, we would agree on the local state of each node. Every time the agreement instances terminate the nodes may evaluate a predicate that can determine whether the past global state was legal. The nodes may then decide whether to reset the non-stabilizing algorithm accordingly.

The next subsection specifies some additional results which can be achieved using this scheme.

Additional Results. The digital clock synchronization algorithm presented here can be quickly transformed into a token circulation protocol in which the token is held in turn by any node for any pre-determined number of rounds and in a pre-determined order. The pre-determined variables are part of the required incorruptible code. E.g. if the token should be passed every k beats, then node p_i, $i = 1 + \frac{DigiClock}{k} \bmod n$ holds the token during rounds $[k \cdot (i-1) + 1, k \cdot i]$. Similarly, it can also produce synchronized pulses which can then be used to produce the self-stabilizing counterpart of general *Byzantine* protocols by using the scheme in [3]. These pulses can be produced by setting *overlap* to be the pulse cycle interval, and issuing a pulse each time $DigiClock = 0$.

Digital Clock vs. Clock Synchronization. In the described algorithm, the non-faulty nodes agree on a common integer value, which is regularly incremented by one. This integer value is considered "the synchronized (digital) clock value". Note that clock values estimating real-time or real-time rate can be achieved in two ways. The first one, is using the presented algorithm to create a new distributed pulse, with a large enough cycle, and using the algorithm presented in [6] to synchronize the clocks. The second, is to adjust the local clock of each node, according to the value of the common integer value, multiplied by the predetermined length of the beat interval.[5] This way, at each beat of the global beat system, the clocks of all the nodes are incremented at a rate estimating real-time.

Future Work. We consider three main points to be interesting for future research.

- Can the tolerance of the algorithm be improved to support $f < \frac{n}{3}$?
- Can the above mechanism be applied in a more general way, leading to a general stabilizer of *Byzantine* tolerant algorithms without using the scheme proposed in [3], which requires pulses that are sufficiently spaced apart?

[5] This value need also be defined as part of the incorruptible code of the nodes.

– What happens if the global beats are received at intervals that are less than the message delay, i.e. common clock beats. Is there an easy solution to achieve synchronized clocks? If yes, can it attain optimal precision like the current solution? If no, is the only option then to synchronize the clocks in a fashion similar to [2,5,6]? i.e. by executing an underlying distributed pulse primitive with pulses that are far enough apart in order to be able to terminate agreement in between. In that case, is there any advantage in having a common source of the clock ticks or is it simply a replacement for the local timers of the nodes?

References

1. A. Arora, S. Dolev, and M.G. Gouda, "*Maintaining digital clocks in step*", Parallel Processing Letters, 1:11-18, 1991.
2. A. Daliot and D. Dolev, "*Self-stabilizing Byzantine Pulse Synchronization* ", Technical Report TR2005-84, Schools of Engineering and Computer Science, The Hebrew University of Jerusalem, August 2005. A revised version appears in http://arxiv.org/abs/cs.DC/0608092
3. A. Daliot and D. Dolev, "*Self-stabilization of Byzantine Protocols*", Proc. of the 7th Symposium on Self-Stabilizing Systems (SSS'05 Barcelona), pp. 48-67, 2005.
4. A. Daliot and D. Dolev, "*Self-stabilizing Byzantine Agreement*", Proc. of Twenty-fifth ACM Symposium on Principles of Distributed Computing (PODC'06), Denver, Colorado, July 2006.
5. A. Daliot, D. Dolev and H. Parnas, "*Self-stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks*", Proc. of the 6th Symposium on Self-Stabilizing Systems (SSS'03 San-Francisco), pp. 32-48, 2003.
6. A. Daliot, D. Dolev and H. Parnas, "*Linear Time Byzantine Self-Stabilizing Clock Synchronization*", Proc. of 7th International Conference on Principles of Distributed Systems (OPODIS'03 La Martinique, France), December, 2003. A corrected version appears in http://arxiv.org/abs/cs.DC/0608096 .
7. D. Dolev, R. Reischuk, H. R. Strong, "*'Eventual' Is Earlier than 'Immediate'*", In Proceedings, 23nd Annual Symposium on Foundations of Computer Science, 196-203, Nov. 1982
8. S. Dolev, "*Possible and Impossible Self-Stabilizing Digital Clock Synchronization in General Graphs*", Journal of Real-Time Systems, no. 12(1), pp. 95-107, 1997.
9. S. Dolev, and J. L. Welch, "*Self-Stabilizing Clock Synchronization in the presence of Byzantine faults*", Journal of the ACM, Vol. 51, Issue 5, pp. 780 - 799, 2004.
10. S. Dolev and J. L. Welch, "*Wait-free clock synchronization*", Algorithmica, 18(4):486-511, 1997.
11. T. Herman, "*Phase clocks for transient fault repair*", IEEE Transactions on Parallel and Distributed Systems, 11(10):1048-1057, 2000.
12. B. Liskov, "*Practical Use of Synchronized Clocks in Distributed Systems*", Proceedings of 10^{th} ACM Symposium on the Principles of Distributed Computing, 1991, pp. 1-9.
13. M. R. Malekpour, and R. Siminiceanu, "*Comments on the "Byzantine Self-Stabilizing Pulse Synchronization" Protocol: Counterexamples*", NASA/TM-2006-213951, February 2006.
14. M. Papatriantafilou, P. Tsigas, "*On Self-Stabilizing Wait-Free Clock Synchronization*", Parallel Processing Letters, 7(3), pages 321-328, 1997.
15. S. Toueg, K. J. Perry, T. K. Srikanth, "*Fast Distributed Agreement*", SIAM Journal on Computing, 16(3):445-457, June 1987.

A $\mathcal{BC}$ Protocol

A typical synchronous *Byzantine* agreement / consensus protocol runs in a fixed number of rounds (Δ) (for practical ones it is about $2f + 4$) rounds (or phases). If such a protocol will be invoked by at least $n - f$ non-faulty nodes, without having in their memory any residue of previous runs of the protocol, it will end up producing a consensus value at all non-faulty nodes. Therefore, it is enough to augment any such protocol with an initial action of resetting all variables used in the protocol when it is invoked.

Our results require that the non-faulty nodes execute a consensus protocol. All synchronous agreement protocols can be converted to consensus. The "Solidarity" requirement poses no problem, since in consensus protocols all nodes exchange values at the first round, and only a value that was sent by $n - f$ nodes (at least $n - 2f$ of them are non-faulty) will be a candidate value at the next phase. It requires setting the default value to $\perp$.

Specifically, the non-self-stabilizing protocols in [7,15] can be transformed to the required $\mathcal{BC}$ protocol. Similarly the self stabilizing protocol in [4] can be simplified to a synchronous protocol satisfying the required properties.

In the context of the current paper each node executes Δ copies of the modules, each copy is separated from the others, therefore, the messages of the non-faulty nodes are separated for each copy. Faulty nodes may behave arbitrarily. Notice that a node that just recovered from a transient fault invokes a clean copy of $\mathcal{BC}$ which it executes correctly, while the other copies may still be affected by the transient fault. Notice that after executing Line 5 of the digital clock synchronization algorithm Digital-SSByz-ClockSync a non-faulty node can distinguish between the Δ copies of $\mathcal{BC}$.