Self-Stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks

Ariel Daliot
School of Engineering and Computer Science
Hebrew University of Jerusalem, Israel
ariel.daliot@intel.com, adaliot@cs.huji.ac.il

Danny Dolev
School of Engineering and Computer Science
Hebrew University of Jerusalem, Israel
dolev@cs.huji.ac.il

Hanna Parnas
Department of Neurobiology and
the Otto Loewi Minerva Center for Cellular and Molecular
Neurobiology, Institute of Life Science,
Hebrew University of Jerusalem, Israel
hanna@vms.huji.ac.il

March 13, 2003

Abstract

We define the "pulse synchronization" problem that requires nodes to achieve tight synchronization of regular pulse events. Pulse-coupled synchronization is a phenomenon displayed by a large variety of distributed biological systems, overcoming the typically large level of noise present. Inspired by such biological models, a robust and self-stabilizing pulse synchronization algorithm for distributed computer systems is presented. The algorithm attains near optimal synchronization while tolerating up to a third of the nodes exhibiting Byzantine behavior concurrently. Pulse synchronization algorithms can be suitable for a variety of distributed tasks that require tight synchronization but which can tolerate a bound variation in the regularity of the synchronized pulse invocations.

Keywords: Self-stabilization,pulse synchronization,biological synchronization,biological oscillators.

1 Introduction

The phenomenon of synchronization is displayed by many biological systems [18]. It presumably plays an important role in these systems. For example, the heart of the lobster is regularly activated by the synchronized firing of four interneurons in the cardiac ganglion [9, 10]. It was concluded that the organism cannot survive if all four interneurons fire out of synchrony for prolonged times [17]. This system inspired the present work. Other examples of biological synchronization include the *malaccae* fireflies in Southeast Asia where thousands of male fireflies congregate in mangrove trees, flashing in synchrony [3]; oscillations of the neurons in the circadian pacemaker, determining the day-night rhythm; crickets that chirp in unison [20]; coordinated mass spawning in corals and even audience clapping together after a "good" performance [16]. Synchronization in these systems is typically attained despite the inherent variations among the participating elements, or the presence of noise from external sources or from participating elements. A generic mathematical model for synchronous firing of biological oscillators based on a model of the human cardiac pacemaker is given in [15]. This model does not account for noise or the inherent difference among biological elements.

In computer science, synchronization is both a goal by itself and a building block for algorithms that solve other problems. In the "Clock Synchronization" problem, it is desired for computers to have their clocks set as close as possible to each other as well as to keep a notion of real time. This means essentially keeping the clocks close to a constant frequency - the real time, in addition to being synchronized to each other [5].

It is desired for algorithms to guarantee correct behavior of the network in face of faults or failing elements, sometimes irrespective of any initial state of the system (self-stabilizing). It has been suggested that similar fault considerations may have been involved in the evolution of distributed biological systems, [17]. In the example of the cardiac ganglion of the lobster, it was concluded that at least four neurons are needed in order to overcome the presence of one faulty neuron, though supposedly one neuron suffices to activate the heart. The cardiac ganglion must be able to adjust the pace of the synchronized firing according to the required heartbeat, up to a certain bound, without loosing the synchrony (e.g. while escaping a predator a higher heartbeat is required – though not too high). Due to the vitality of this network, it is presumably optimized for fault tolerance, self-stabilization, tight synchronization and for fast re-synchronization.

The apparent resemblance of the synchronization and fault tolerance requirements of biological networks and distributed computer networks makes it very appealing to infer from models of biological systems onto the design of distributed algorithms in computer science. Especially when assuming that distributed biological networks have evolved over time to particularly tolerate inherent in-homogeneity of the cells, noise and cell death. In the current paper, we show that in spite of obvious differences, a biological fault tolerant synchronization model ([17]) can inspire a novel solution to an apparently similar problem in computer science.

We propose the problem definition of "pulse synchronization" for distributed computer systems, in which all the elements are required to invoke some regular pulse (or perform a "task") in tight synchrony, but allow for some variation from complete regularity. Though nodes need to invoke the pulses synchronously, there is a limit on how frequent it is allowed to be invoked (similar to the linear envelope clock synchronization limitation). We present a novel algorithm in the settings of self-stabilizing distributed algorithms, instructing the nodes how and when to fire in order to meet the synchronization requirements of "pulse synchronization". The core elements of the algorithm are analogous to the neurobiological principles of *endogenous (self generated) periodic spiking, summa-*

tion and *time dependent refractoriness*. The basic algorithm is quite simple: every node invokes a pulse regularly and sends a message upon invoking it (*endogenous periodic spiking*). The node sums messages received in some "window of time" (*summation*) and compares this to the continuously decreasing time dependent firing threshold for invoking the pulse (*time dependent refractoriness*). The node fires when the counter of the summed messages crosses the current threshold level, and resets its cycle.

The algorithm performs correctly in the presence of nodes that can behave in a completely arbitrary ("Byzantine") manner, as long as less than a third of all the nodes display Byzantine behavior concurrently. It ensures a tight synchronization of the pulses of all correct nodes, while not using any central clock or global pulse. We assume the physical network allows for a broadcast environment and has a bounded delay on message transmission. The algorithm may not reach its goal as long as these limitations are violated or the network graph is severely disconnected. These limitations will be referred to as *the network's assumption boundaries*. The algorithm is self-stabilizing and thus copes with a more severe fault model than the traditional Byzantine fault models. Traditional distributed algorithms assume the system is always within the predefined assumption boundaries. Moreover, many algorithms, particularly non-stabilizing clock synchronization algorithms, also make strong assumptions on the initial state of the nodes (such as assuming all clocks are initially synchronized, c.f. [5]). But what happens if the system is temporarily thrown out of the assumption boundaries? (E.g. more than one third of the nodes are faulty or messages of correct nodes get lost). When the system eventually returns to the predefined assumptions it may be in any arbitrary state. A self-stabilizing protocol converges to its goal once the system is back within the assumption boundaries. For our protocol, once this happens, regardless of the state of the system, tight synchronization is achieved within finite time. It overcomes transient, permanent and intermittent faults, though for convergence we assume that there is a window of time during which no recovery of nodes takes place.

Our algorithm is uniform, all nodes execute an identical algorithm. It does not suffer from communication deadlock, as can happen in message-driven algorithms ([2]), since the nodes have a time-dependent state change, at the end of which they fire endogenously. It is fault-containing in the sense that the convergence time depends on the number of faulty nodes, f, and not on the network size (In Section 5 we discuss a relationship between the cycle length and the number of faults, imposed by our algorithm). The faulty nodes cannot ruin the synchronization; in the worst case, they can slow down the convergence and speed up the synchronized firing frequency up to a certain bound. The time complexity achieved is $O(f)$ cycles with a near optimal synchronization tightness of d (end to end network and processing delay). Comparatively, the time complexity of the digital clock synchronization problem in a similar model presented in [8] is exponential in the network size. The synchronization tightness achieved there depends on the network size.

2 Assumptions, definitions and specifications of the algorithm

The environment is a network of processors (nodes) that regularly invoke pulses, ideally every *cycle* time units. The invocation of the pulse is expressed by sending a message via a broadcast media; this is also referred to as **firing**.

In our environment, individual nodes have no access to a central clock or global pulse system. Moreover, the hardware clocks of **correct nodes** have a bounded drift rate, ρ, from real time. The communication network does not guarantee any order on messages, though there is a bound on

Slowest node	Real time	Fastest node
$x(1-\rho)$	x	$x(1+\rho)$
x	$\frac{x}{1-\rho}$	$\frac{x(1+\rho)}{1-\rho}$
$\frac{x(1-\rho)}{1+\rho}$	$\frac{x}{1+\rho}$	x

Table 1: Time count conversion table between nodes displaying the upper and lower bounds of the allowed clock skews (fastest and slowest nodes respectively). For example, suppose the first row corresponds to a time frame of 1000 real time units. A hardware clock whose skew is at the lower end (slowest) of the allowed range will count only $1000000(1-10^{-6}) = 999999$ time units, while a hardware clock with the highest acceptable skew (fastest) will count $1000000(1+10^{-6}) = 1000001$ time units accordingly.

message delivery time once the network stabilizes.

The network and/or the nodes can behave arbitrarily, though eventually the network stabilizes to within the defined assumption boundaries in which at most f nodes may behave arbitrarily. The **network assumption boundaries** are:

1. Multicast message passing allowing for authentication of the sender.
2. At most f of the nodes are faulty.
3. The subnetwork of correct nodes satisfies:
   (a) There is an upper bound δ on the physical end-to-end network delay.
   (b) Any message sent or received by any correct node will eventually reach every correct node within δ time units.

A node is **correct** at times that it complies with the following four conditions:

1. Obeys a global constant $0 < \rho_{glob} << 1$, such that for every Newtonian time interval $[u, v]$, $(1-\rho_{glob})(v-u) \leq$ 'physical clock' $\leq (1+\rho_{glob})(v-u)$. Hereafter ρ_{glob} is denoted by ρ ($\rho \approx 10^{-6}$ in modern computers).
2. Operates according to an instructed protocol.
3. Accepts and sends only multicast messages.

Thus, a node is considered **faulty** when it violates one or more of the above. A faulty node can recover from its faulty behavior. The recovery is not necessarily immediate, but rather can take a certain amount of time until which the node is still considered faulty. We count the recovering nodes as part of the faulty nodes during a time window of length *cycle*. Thus, within the network assumption boundaries there can be at most f faulty and recovering nodes in every *cycle* time window. To overcome intermittent faults we need to assume some bound on how frequently the adversary can fail a recovered node, since, over time, all nodes may repeatedly fail and recover. We assume that eventually there is a window of at least $2(2f+1)$ cycles within which there is no recovery of nodes, which is proved in Theorem 3 to allow the system to converge.

Table 1 displays the actual time differences counted by clocks exhibiting the allowed extremes of the clock skew.

Basic notations:
We use the following notations to define the quality of the solution, though nodes do not need to maintain them as variables.

- σ represents the target upper bound on the real time between the invocations of the pulses of different correct nodes (*tightness of synchronization*).
- $\Psi_p(t)$ is the number of endogenous (self generated) pulses node p would have invoked since its last recovery time to time t, had it been alone in the system.
- $\Phi_p(t)$ is the effective (actual) number of pulses a correct node p invoked since its last recovery time to time t.
- Let $a, b, g, h \in R^+$ be constants that define (see Figure 1) the linear bounds on the ratio between $\Psi_p(t)$ and $\Phi_p(t)$.
- $\phi_i \in R^+ \cup \{\infty\}$ denotes the real time that has passed since the last time node p_i invoked a pulse. For a node, p_j, that has not fired since initialization of the system, $\phi_j \equiv \infty$.
- $d \equiv \delta + \pi$, where π is the upper bound on the message processing time.

Thus, d is an upper bound on the elapsed real time from the arrival of a message at a correct node and until a consequent message that is sent arrives at all other correct receiving nodes.

Basic definitions:

- The **state** of the system at time t is given by: $State \equiv (t, \phi_0, \ldots, \phi_{n-1})$.
- Let G be the set of all possible system states of a system S.
- A set of nodes, N, are called **synchronized** at time t if $\forall p_i, p_j \in N$,

$$\phi_i, \phi_j \leq \frac{cycle}{1-\rho}, \text{ and } |\phi_i - \phi_j| \leq \sigma \text{ or } \frac{cycle}{1-\rho} - \sigma \leq |\phi_i - \phi_j| \leq \frac{cycle}{1-\rho}.$$

  Note that every correct node is always synchronized with itself. If $\forall p_i, p_j \in N, \phi_i, \phi_j \leq \frac{cycle}{1-\rho}, |\phi_i - \phi_j| \leq \sigma$ then we say that N is **strongly synchronized.**
- $s \in G$ is a **synchronized state** of the system at time t if the set of correct nodes are strongly synchronized at some time t_{syn} in the interval $[t, t+\sigma]$.
- $s \in G$ is a **sub-synchronized state** of the set of nodes N at time t, if the set of correct nodes in N are strongly synchronized at some time t_{syn} in the interval $[t, t+\sigma]$.

Note that when $||N|| = n$ for the set of nodes N, then the two definitions coincide. Observe that the definition of synchronized nodes is not transitive, and therefore the definition of a sub-synchronized state is not transitive.

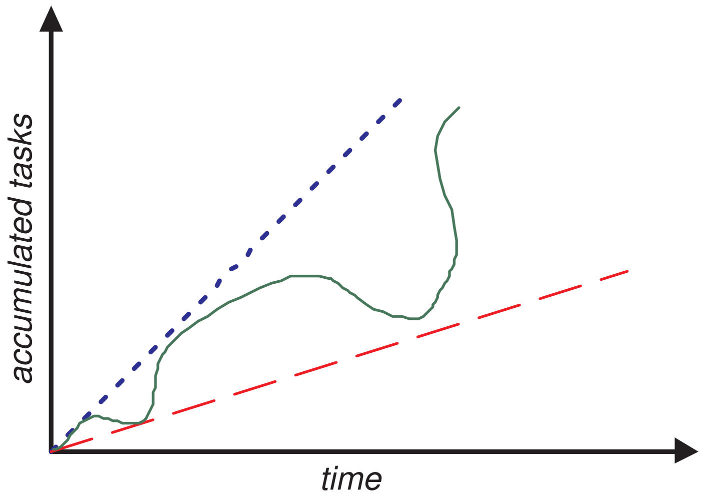

Figure 1: Schematic illustration of the linear envelope of firing frequency showing the allowed upper and lower limits of the effective number of tasks performed over time. Upper bound on number of events, $g[\Psi(time)]+h$; Lower bound on number of events, $a[\Psi(time)]+b$; Actual number of events, $\Phi(time)$.

The Self-Stabilizing Pulse Synchronization Problem

As long as the system is within its assumption boundaries:

Convergence: Starting from an arbitrary state, s, the system reaches a synchronized state after a finite number of steps.

Closure: If s is a synchronized state of the system then all subsequent system states are synchronized states, as long as the following conditions are satisfied:

1. ≪Linear Envelope≫: for every correct node, p, and for every time t, $a[\Psi_p(t)] + b \leq \Phi_p(t) \leq g[\Psi_p(t)] + h$.
2. No faulty node recovers.

Note that the linear envelope (see Figure 1) resembles, in a sense, the linear envelope requirement in the clock synchronization problem.

3 The "Pulse synchronization" Algorithm

The algorithm will follow its neurobiological analog. Each node sums the pulses that it learns about during a recent time window. If this sum (the Counter) crosses the current (time-dependent) threshold, then the node will fire. The refractory function describes the time dependency of the threshold. At threshold level 0 the node fires **endogenously,** irrespective of any inputs.

The refractory function
Recall that the cycle is the predefined time a correct node counts between two endogenous pulses.

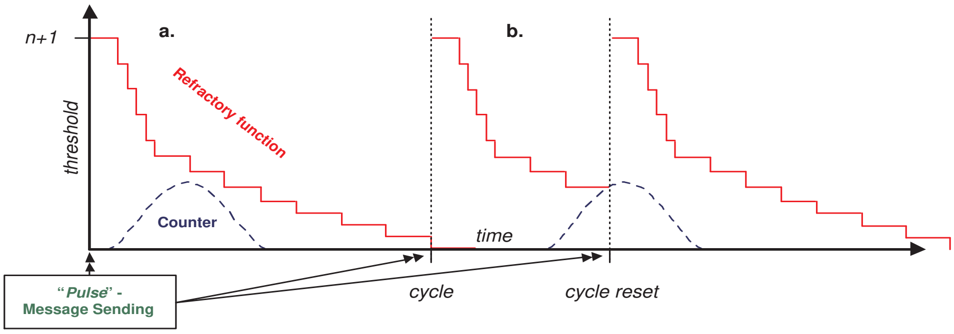

Figure 2: Schematic example of the algorithm's mode of operation. (a.) A node fires endogenously if threshold=0 is reached. The counter of the summed messages does not cross the refractory function during the cycle, letting the threshold reach zero and consequently the node fires endogenously. (b.) The Counter of the summed messages increases sufficiently to cross the current refractory function, consequently the node fires and a new cycle of the node is initiated.

The refractory function R, is a step function. R_i is the time length of threshold level i. $R_i \in R^+$, $i = 1 \ldots n+1$, where, $\sum_{i=1}^{n+1} R_i \equiv cycle$. R_{n+1} is called the **absolute refractory period** of the cycle. Using the neurobiological analogy, this is the first period after a node fires; the node never fires within an absolute refractory period. We assume that $cycle \gg \sigma$ and require that $R_{n+1} > 2\sigma$ (satisfied by Restriction 5 presented later). We denote $R^p(t)$ the threshold level of node p at time t or $R(t)$ if it is clear what node is referred to. If a node fires at time t, then R is **reset** to $R(t) = n+1$.

The messages

The content of a message is a Counter - the number of recent inputs received that caused the sending node to fire. A **k-sized input** is a set of k such messages that are received within a certain time window (whose exact properties is described in the appendix) that causes the value of the Counter of the receiver to be at least k.

The Summation algorithm

A full description of the procedures used by the Summation Algorithm is provided in the Appendix. The Summation Algorithm is comprised of the following components:

The TIMELINESS procedure determines if the Counter contained in the message seems "reasonable" (timely). The bound on message transmission and processing time among correct nodes allows a node to estimate whether the content of a message it receives is plausible and therefore timely. If it is not then the node tables it for possible future use and after a certain time decays it. The MAKE-ACCOUNTABLE procedure determines by how much to increment the Counter following the arrival of a timely message. The node can take into account tabled messages, if necessary, for consistency reasons. The PRUNE procedure decays old messages and tables (currently) irrelevant messages.

Summation((*new message* M_p *from node* p *that arrived at time* t_{event}) or (*change in* R))
  if (*new message*) then
    if (TIMELINESS(M_p, t_{event})) then
      MAKE-ACCOUNTABLE(M_p);
      PRUNE(t);
  if (*an event at time* t_{event} *implied "R has changed"*) then

PRUNE(t).

The event driven self-stabilizing algorithm

Pulse-Synchronization$(n, cycle)$

INIT: if (n and $cycle$ are valid) then
  R := as determined by Eq. 13;
  reset R;
  set Counter $= 0$;
 else *exit;*
WAIT-FOREVER-LOOP:
 if (*at time* t_{event} *a new message,* M_p *arrives from a node* p or *change in* R) then
  SUMMATION((M_p, t_{event}) or (*change in* R));
  if ($Counter \geq R(t)$) then
   SIGN&SEND; // Invocation of the Pulse;
   reset R.

4 Solving the "Pulse synchronization" problem

This section introduces notations and procedures for proof purposes only. To ease following the dynamics in the n-dimensional state space we show that the state space can be divided into a small number of stable fixed points ("synchronized clusters") such that the state of each individual node is attracted to one of the stable fixed points. We show that there are always at least two of these fixed points that are situated in the basins of attraction ("absorption distance") of each other. Following the dynamics of these attractors, we show that eventually the states of all nodes settle in a limit cycle in the basin of one attractor. We prove that within a finite number of cycles, for some range of parameters, the synchronized clusters repeatedly absorb to form a set of nodes in a sub-synchronized state, until a synchronized state of the system is reached (*Convergence*). In Lemma 4.2, we prove that once the system is in a synchronized state it remains as such (*Closure*). The synchronization tightness, σ, of our algorithm, equals d.

"First node" in a synchronized set of nodes Γ, is a node in the subset of nodes that "fire first" in Γ that satisfies:

$$\text{“First node in } \Gamma\text{”} = \begin{cases} \min\{i | i \in \max\{\phi_i | \text{node } i \in \Gamma, \phi_i \leq \sigma\}\} & \exists i \in \Gamma \, \text{s.t.} \, \phi_i \leq \sigma \\ \min\{i | i \in \max\{\phi_i | \text{node } i \in \Gamma\}\} & \text{otherwise.} \end{cases}$$

Equivalently, we define "**last node**":

$$\text{“Last node in } \Gamma\text{”} = \begin{cases} \max\{i | i \in \min\{\phi_i | \text{node } i \in \Gamma, \phi_i > \sigma\}\} & \exists i \in \Gamma \, \text{s.t.} \, \phi_i > \sigma \\ \max\{i | i \in \min\{\phi_i | \text{node } i \in \Gamma\}\} & \text{otherwise.} \end{cases}$$

Synchronized Clusters
At a given time t the nodes are divided into disjoint **synchronized clusters** (or just **clusters**) in the following way:

1. Assign the maximal set of synchronized nodes at time t as a synchronized cluster.
2. Assign the second maximal set of synchronized nodes that are not part of the first cluster as a synchronized cluster. In case there are several maximal sets choose the set that is harboring the first node of the unified set of all these maximal sets.
3. Continue until all nodes are exclusively assigned to some synchronized cluster.

The cluster with the node having the largest (necessarily finite) phase among all the nodes is designated C_1. The rest of the clusters are enumerated inversely to the phase of their first node, thus C_c is the cluster whose first node has the lowest phase (besides perhaps C_1). The first cluster is treated separately because it is the only cluster whose nodes' actual phase differences can be larger than σ as it can contain nodes that have just fired and nodes just about to fire.

The clustering is done only for the proof. It doesn't actually affect the protocol or the behavior of the nodes. We can assume that at any moment we fixate a clustering of the system and look at its development over time. Once such a clustering is fixated we remove from it nodes that happen to fail and ignore forthcoming recovering nodes. Our proof is based on the observation that eventually we reach a time window within which the total number of faulty nodes is bounded by f with no node recovery and during that window the whole system converges.

We will frequently use the following definitions and notations:

- C_i – synchronized cluster number i.
- n_i – cardinality of C_i (i.e. number of correct nodes associated with synchronized cluster C_i).
- c – current number of synchronized clusters in the current state; $c \geq 1$.
- $dist(a, b) \equiv |\phi_a - \phi_b|$ is the **phase difference** between nodes a and b.
- $dist(C_i, C_j) \equiv dist$ (first node in cluster C_i, first node in cluster C_j).

If $\phi_{\textit{first node in cluster}\, c_i} \geq \phi_{\textit{first node in cluster}\, c_j}$ then we say that cluster C_i **precedes** cluster C_j. If there exists no other cluster C_r, such that

$$\phi_{\textit{first node in cluster}\, c_i} \geq \phi_{\textit{first node in cluster}\, c_r} \geq \phi_{\textit{first node in cluster}\, c_j}$$

then we say that the clusters C_i and C_j are **adjacent.**

Corollary 4.1 *If we let $(C_i, C_{(i+1) \pmod c})$ denote adjacent clusters, then*

$$\sum_{i=1}^{c} dist(C_i, C_{(i+1) \pmod c}) \leq (1 + \rho)\, cycle. \tag{1}$$

We say that two clusters, C_i and C_j, have **absorbed** if their union comprises a set of nodes in a sub-synchronized state, and that originally were not in a sub-synchronized state. If a node in C_j fires due to a message received from a node in C_i, then, as will be shown in Theorem 1, the inevitable result is that their two clusters absorb. The course of action from the arrival of the message at a node in C_j until C_j has absorbed with C_i is referred to as the **absorption** of C_j by C_i.

The proof that the synchronization protocol satisfies the requirements of the problem follows the steps below:

In Subsection 4.1 (Lemma 4.3), we show that the nodes of a synchronized cluster always stay in a sub-synchronized state. In Theorem 1 we show that by imposing several restrictions on the various parameters we can explicitly determine a threshold value $ad(C_i)$ that has the property that if for two synchronized clusters C_i and C_j, $dist(C_i, C_j) \leq ad(C_i)$ then C_i **absorbs** C_j. We will call that value the "absorption distance" of C_i.

In Subsection 4.2, we specify the requirements for a refractory function R that ensure the continuous existence of a pair of synchronized clusters within the absorbance distance of each other, and hence absorb. Thus, iteratively, all synchronized clusters will eventually absorb to form a single synchronized cluster, which stays in a sub-synchronized state. Thus, the Closure and Convergence requirements are satisfied.

In Subsection 4.3, we show that the refractory function of our algorithm satisfies the requirements and therefore comprises a solution to the pulse synchronization problem.

4.1 Determining the Absorption Distance

We define five restrictions on the system's parameters and protocols that ensure the correct behavior of the algorithm.

Restriction 1: The Summation algorithm counts input messages from other nodes such that: Following the arrival of a message from a correct node, every correct receiving node's Counter is incremented to hold a value greater than the Counter in the message.

Restriction 2: R_i is a monotonic decreasing function of i, $R_i \geq R_{i+1}, i = 1 \ldots n-1$.

Restriction 3: $R_i > 3d$, $i = 1 \ldots n-f-1$.

Restriction 4: $R_i > \sigma(1+\rho) + \frac{\rho}{1+\rho}\sum_{j=1}^{n+1} R_j, i = 1 \ldots n$.

Restriction 5: $R_{n+1} \geq 2d(1+\rho)\frac{(\frac{1+\rho}{1-\rho})^{n+3}-1}{(\frac{1+\rho}{1-\rho})-1}$.

We first show that the synchronized cluster state is a stable fixed point in state space.

Lemma 4.2 *A set of correct nodes that are in a sub-synchronized state at some time t', remain in a sub-synchronized state as long as the network remains within its assumption boundaries.*

Proof: In accordance with the definition of the sub-synchronized state this set of nodes will be strongly synchronized at some time t_{syn} in the interval $[t', t'+\sigma]$. The set will stay strongly synchronized (for at least $\frac{cycle}{1-\rho}$) and thus in a sub-synchronized state as long as no node in the set fires. This is because the phase difference between nodes (measured on a real time clock) does not change as long as none of them fires. We therefore turn our attention to the first occasion after t_{syn} in which a node from the set fires. Let us examine the extreme case seen in Figure 3 with a sub-synchronized set consisting of at least two nodes at the maximal allowed phase difference; that is to say that at time t_{syn}, $dist(\phi_{first-node}, \phi_{last-node}) = \sigma$, where $\phi_{first-node}, \phi_{last-node} \leq \frac{cycle}{1-\rho}$. Further assume that the first node in the set receives a k-sized input ($0 \leq k \leq n-1$) at some time $t'' \geq t_{syn}$ at the very beginning of its threshold level k, and is also the first node in the cluster to fire after time t_{syn}. We will show that the rest of the nodes in the set will fire within the interval $[t'', t''+\sigma]$ and thus the set remains in a sub-synchronized state.

Restriction 1 ensures that the last node's Counter will read at least $k+1$ subsequently to the arrival and assessment of the first node's fire, since its Counter should be at least the first node's Counter plus 1. The proof of the lemma will be done by showing that right after the assessment of the first node's fire, the last node cannot be at a threshold higher than $k+1$ and thus will necessarily fire.

The proof is divided into the following steps:

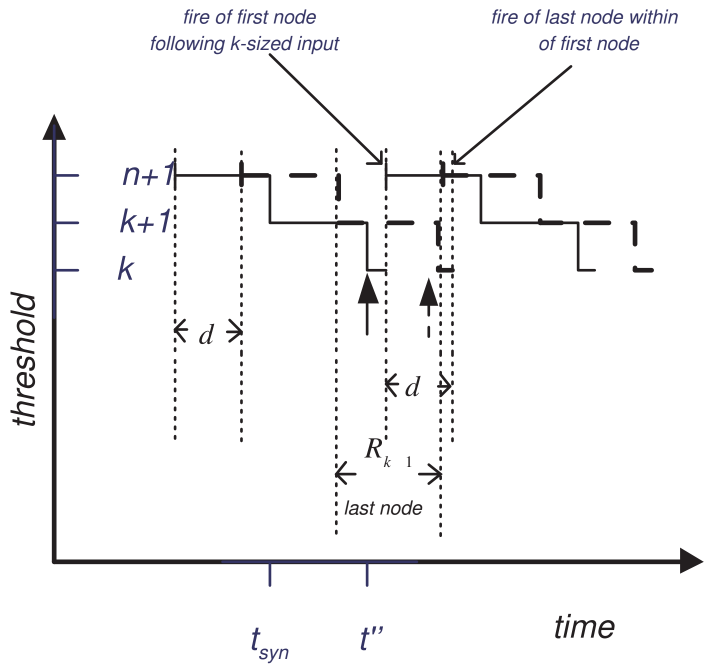

Figure 3: A schematic drawing of the scenario in which Lemma 4.2 is proven. —— denotes the refractory function of the first node. - - - denotes the refractory function of the last node at a shift of $d = \sigma$. The event of arrival of a k sized input at the first node is symbolized by the solid up arrow. The last node receives the resultant $k+1$ sized input at the time pointed to by the dashed arrow, at its threshold level $k+1$. R_{k+1} is the time length of the last node's threshold level $k+1$.

1. Show that if the first node is at threshold level k then the last node is at most at threshold level $k+1$.
2. Show that if the first node receives a k-sized input and fires then the last node will fire consequently.
3. Show that the last node fires within a s time window of the first node, and as a result, the new distance between the first and last node is less or equal σ.

The same applies to any intermediate node in the synchronized cluster and thus the cluster remains in a sub-synchronized state.

Step 1: Assume the first node to be the fastest permissible node and the last one the slowest. Hence, when the first node's threshold level k commences

$$\frac{1}{1+\rho}\sum_{i=k+1}^{n+1} R_i \tag{2}$$

real time units actually passed since it last fired. The last node "counted" this period as:

$$\frac{1-\rho}{1+\rho}\sum_{i=k+1}^{n+1} R_i \tag{3}$$

The last node, moreover, still has to count, at most, additional $\sigma(1+\rho)$ time units to reach the same threshold level (max. phase difference of sub-synchronized nodes as counted by the slowest node). The maximum time difference between the time the first node starts its threshold level k till the last node starts its own threshold level k as counted by the last node is therefore $\sigma(1+\rho)$ plus the difference Eq. 2 – Eq. 3, yielding

$$\sigma(1+\rho) + \frac{\rho}{1+\rho} \sum_{i=k+1}^{n+1} R_i \tag{4}$$

Restriction 4 ensures that R_{k+1} is greater than Eq. 4 for $0 \leq k \leq n-1$; thus when the first node commences threshold level k the last node can be at most at threshold level $k+1$.

Step 2: Let the first node receive a k-sized input at time t'' at threshold level k and fire consequently. In case that the last node receives almost immediately the first node's fire (which is a $(k+1)$-sized input, following Restriction 1), it will be at most at threshold level $k+1$ (following step 1) and will therefore fire. All the more so if the first node's fire is received later, since the threshold level can only decrease in time before a node fires.

Step 3: We now need to estimate the new distance between the first and last node in order to show that they are still synchronized. The last node assesses the first node's fire within d real time units after the first node sent its message (per definition of d). This yields a distance of $d(1-\rho)$ as seen by the last node, which equals the maximal allowed real time distance, d $(=\sigma)$, between strongly synchronized nodes and thus they are in a sub-synchronized state at time t''. □

Lemma 4.3 *The set of nodes comprising a synchronized cluster at some time t remain in a sub-synchronized state as long as the network remains within its assumption boundaries.*

Proof: If the cluster is strongly synchronized at time t then the set of nodes are in a sub-synchronized state at time t. Therefore let the cluster be synchronized but not strongly synchronized at time t. In accordance with the definition of a synchronized set of nodes the nodes were strongly synchronized at time $t - \phi_{first-node}$ and were as such in a sub-synchronized state at time $t - \phi_{first-node}$. Following Lemma 4.2 the set of nodes are thus in a sub-synchronized state at time t and remain as such as long as the network is within its assumption boundaries. □

Corollary 4.4 *A synchronized cluster encompassing all the correct nodes implies the system is in a synchronized state, satisfying the objective of the Pulse synchronization problem.*

Lemma 4.5 *Let $p, q \in C_1$ and $r \in C_2$, denote three correct nodes belonging to two different synchronized clusters. Following the arrival and assessment of p's and q's fires, both will be accounted for in the Counter of r.*

The proof of Lemma 4.5 is given in the appendix.

Following Lemma 4.5, we will refer to the **fire of a cluster** instead of referring to the sum of the fires of the individual nodes in the cluster. We now explicitly determine the basin of attraction (absorption distance) of the synchronized cluster states in state space.

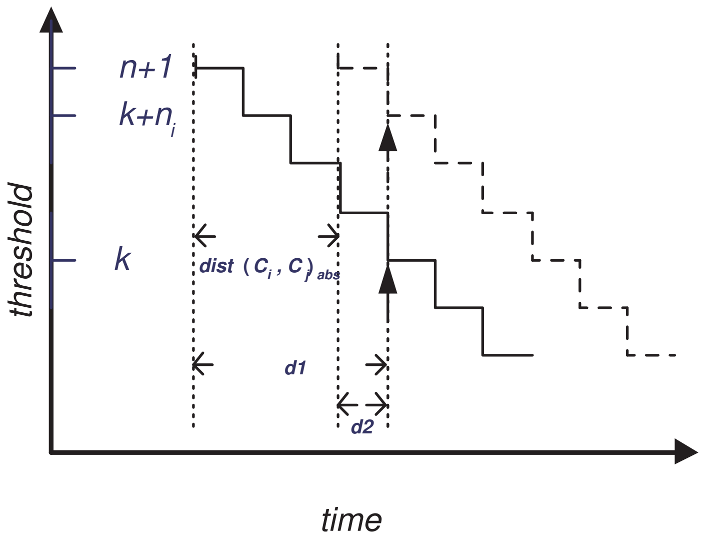

Figure 4: A schematic drawing of the rationale behind Theorem 1. - - - - denotes the refractory function of the first node of C_i. - - - - denotes the refractory function of the first node of C_j. The event of arrival of a k-sized input at the first node of C_i at the beginning of threshold level k is symbolized by the solid up-arrow. The first node of C_j receives the resultant $k+n_i$-sized input at the time pointed to by the dashed up-arrow, at the beginning of threshold level $k+n_i$.

Theorem 1 *Given two synchronized clusters, C_i preceding C_j, Restriction 1 through Restriction 4 imply that if C_i fires due to a k-sized input ($0 \le k \le f$)*[1] *and* **prior** *to the firing of C_i, $dist(C_i, C_j) \le \frac{1}{1-\rho} \sum_{g=f+1}^{f+n_i} R_g - \frac{2\rho}{1-\rho^2} \sum_{g=f+1}^{n+1} R_g$ then C_i absorbs C_j.*

Proof: Let us examine the case in which C_i fires first and that C_j doesn't fire until it receives C_i 's fire. Moreover, assume that the first node in C_i receives a k-sized input ($0 \le k \le f$) at the beginning of threshold level k (see Figure 4). Additionally, in the worst case, assume the first node in C_j receives this cumulative $k+n_i$ sized input almost immediately and at the very beginning of threshold level $k+n_i$. Lemma 4.2 and Restriction 1 guarantee that the rest of the nodes in the clusters will follow their respective first ones within σ. This scenario defines an initial distance, $dist'(C_i, C_j)$, between C_i and C_j within which the fire of C_i causes C_j to fire. A larger initial $dist(C_i, C_j)$ gives rise to the possibility that the cumulative $k+n_i$ sized input will arrive at C_j before the beginning of threshold level $k+n_i$, and hence C_j will not fire. On the other hand if the initial dist(C_i ,C_j) is smaller than dist'(C_i ,C_j) then the $k+n_i$ sized input would have arrived at a later time during threshold level $k+n_i$ or even at a lower threshold level, necessarily causing C_j to fire. If the $k+n_i$ sized input arrives at a later time then the first node in C_j will necessarily be at a later stage of threshold level $k+n_i$ or even at a lower threshold level, again, necessarily causing firing of C_j . This implies that for a given set of parameters: C_i , C_j and k, there exists a distance, $dist(C_i, C_j)_{abs}$, beneath which the resultant fire of C_j is assured and above it firing of

[1]It is sufficient to limit the discussion to the case in which Ci receives a $k \le f$-sized ≪external≫ (faulty) input even though in the "real life" scenario C_i can by all accounts receive larger inputs. The justification for this assumption is that a greater than f sized input can actually be looked upon as though the f faulty nodes fired along with some cluster C_i, yielding an $f+n_i$-sized input. We therefore assume the discussed clusters are the clusters firing with or immediately following the k-sized ≪external≫ input, where $k \le f$. The same reasoning is behind the consistency of the algorithm in case all/some of the faulty nodes actually behave correctly. Hence we assume $0 \le k \le f$.

C_j is not assured to happen. We will now calculate the actual value of $dist(C_i, C_j)_{abs}$, assuming the nodes of C_i are fast and the nodes of C_j are slow. Thus, should the nodes of C_j be fast, then $dist(C_i, C_j)_{abs}$ will all the more so ensure the fire of C_j . Using the notation of Figure 4,

$$\begin{aligned}
&dist(C_i, C_j)_{abs} = d1 - d2 = \\
&\quad \frac{1}{1+\rho}\sum_{g=k+1}^{n+1} R_g - \frac{1}{1-\rho}\sum_{g=k+1+n_i}^{n+1} R_g = \frac{1}{1+\rho}\sum_{g=k+1}^{k+n_i} R_g + (\frac{1}{1+\rho} - \frac{1}{1-\rho})\sum_{g=k+1+n_i}^{n+1} R_g = \\
&\quad (\frac{1}{1+\rho} - (\frac{1}{1+\rho} - \frac{1}{1-\rho}))\sum_{g=k+1}^{k+n_i} R_g + (\frac{1}{1+\rho} - \frac{1}{1-\rho})\sum_{g=k+1}^{n+1} R_g = \\
&\quad \frac{1}{1-\rho}\sum_{g=k+1}^{k+n_i} R_g + (\frac{1}{1+\rho} - \frac{1}{1-\rho})\sum_{g=k+1}^{n+1} R_g = \\
&\quad \frac{1}{1-\rho}\sum_{g=k+1}^{k+n_i} R_g + \frac{2\rho}{1-\rho^2}\sum_{g=k+1}^{n+1} R_g. \qquad (5)
\end{aligned}$$

We now show that Eq. 5 reaches a minimum over k at $k = f$.

$$\begin{aligned}
&(\frac{1}{1-\rho}\sum_{g=k+1}^{k+n_i} R_g - \frac{2\rho}{1-\rho^2}\sum_{g=k+1}^{n+1} R_g) - (\frac{1}{1-\rho}\sum_{g=k+2}^{k+1+n_i} R_g - \frac{2\rho}{1-\rho^2}\sum_{g=k+2}^{n+1} R_g) = \\
&\quad \frac{1}{1-\rho}(R_{k+1} - R_{k+1+n_i}) + \frac{2\rho}{1-\rho^2} R_{k+1} > 0. \qquad (6)
\end{aligned}$$

Due to Restriction 2, Eq. 6 is therefore greater than zero and hence Eq. 5 monotonically decreases in k. Thus, Eq. 5 has a minimal value at $k = f$.

$ad(C_i)$ therefore defines to:

$$ad(C_i) \equiv \min_k \left(\frac{1}{1-\rho}\sum_{g=k+1}^{k+n_i} R_g - \frac{2\rho}{1-\rho^2}\sum_{g=k+1}^{n+1} R_g \right) = \frac{1}{1-\rho}\sum_{g=f+1}^{k+n_i} R_g - \frac{2\rho}{1-\rho^2}\sum_{g=f+1}^{n+1} R_g. \qquad (7)$$

We will complete the proof by showing that after C_i causes C_j to fire, the two clusters actually absorb. First, we need to estimate the maximum distance between the two synchronized clusters following the fire of C_j . The first node in C_j can fire at the latest upon receiving and assessing the message of the last node in C_i . More precisely, fire at the latest d real time units following the fire of the last node in C_i , yielding a $dist(C_i, C_j)$ of at most $2d$ regardless of the previous $dist(C_i, C_j)$, n_i, k and n_j. The last node of C_j is at most at a distance of d from the first node of C_j therefore making the maximal distance between the first node of C_i and the last node of C_j equal $3d$. Restriction 3, $R_i > 3d$, $i = 1 \ldots n - f - 1$, ensures that by the next cycle all these nodes fire at most after $d(1+\rho)$ ($= \sigma$ on a real time clock) time units have passed on their clocks subsequent to the fire of C_i 's first node. As a result, the two clusters reach a sub-synchronized state and have thus absorbed, within at most two cycles. Let us now examine the case that C_j is the one to fire first due to the k-sized input, received at a time t_k, at the beginning of threshold level k, before the k-sized input arrives at C_i and that C_i doesn't fire prior to receiving the resulting message from C_j . C_i precedes C_j , meaning $\phi_{\text{first node in cluster}_{C_i}} \geq \phi_{\text{first node in cluster}_{C_j}}$. Let us look at the case in which

$\phi_{\text{first node in cluster}c_i} - \phi_{\text{first node in cluster}c_j} \leq \sigma$, thus they are synchronized just at time t_k and are in a sub-synchronized state at time t_k. Lemma 4.2 thus ensures the first node in C_j will cause the first node in C_i to fire. The same arguments as in the previous case now provide for the absorption of C_i by C_j . Let us now look at the case in which $\phi_{\text{first node in cluster}c_i} - \phi_{\text{first node in cluster}c_j} > \sigma$. $R^{\text{first node in}c_i}(t_k)$ in the current case is smaller than or equal to $R^{\text{first node in}c_i}(t_k)$ of the previous case (the synchronized case). Therefore, if in the synchronized case Lemma 4.2 ensures that the fire of the first node in C_i will follow the fire of the first node in C_j then all the more so in this case.

To conclude the proof we are left with the case that both fire before each one receives the input from the other cluster. In this case, the resulting distance following the fire is even smaller and the two clusters will end up absorbing. This concludes the proof. □

4.2 The pivotal role of the clusters and their absorption ability

We show below that in any state that is not a synchronized state of the system, there are at least two synchronized clusters that are within absorbance distance of each other, and subsequently they absorb. Consequently, the number of clusters will decrease until a synchronized state is reached, $c \xrightarrow{t} 1$. This is done by employing the following two steps:

Step 1: For two synchronized clusters, C_i preceding C_j, Restrictions 1-4 imply that if C_i fires due to a k-sized input $(0 \leq k \leq n-1)$ at a time t_k, and $R^{\textit{first-node-in}c_j}(t_k) \leq n_i$, then consequently C_j will fire by the time it will assess the fire of C_i. Thus, if $dist(C_i, C_j) \leq \frac{1}{1-\rho}\sum_{g=n_i+1}^{n+1} R_g$ at the **time of the fire** of the first node in C_i (thus $\phi_{\textit{first node in}c_i} = 0$), then C_i absorbs C_j.

The reason is that Restriction 1-4 ensure that C_j will count C_i's fire at least as a n_i-sized input, irrespective of k. At this distance, at the time of C_i's fire, assuming the nodes of C_j are slow, the first node in cluster C_j will have counted $(1-\rho)\,(\frac{1}{1-\rho}\sum_{g=n_i+1}^{n+1} R_g) = \sum_{g=n_i+1}^{n+1} R_g$ time units; consequently $R^{\textit{first-node-in}c_j}(t_k) \leq n_i$. Therefore, following Restriction 1, C_j will fire as well. All the more so, if the nodes of C_j are fast or fired earlier (though after the firing of C_i).

Define the **listening-distance** of a cluster C_i, $ld(C_i)$, by the sum $\frac{1}{1-\rho}\sum_{g=n_i+1}^{n+1} R_g$. Because the listening-distance relates to the **time of the fire** of the preceding cluster and not to a time prior to the fire, then we want to define the distance between the two clusters as the real time that would elapse until the latter cluster would have fired endogenously. Thus, we will use in the actual constraints the complementary measure

$$\overline{ld(C_i)} \equiv \frac{1}{1-\rho}\,cycle - ld(C_i) = \frac{1}{1-\rho}\,(cycle - \sum_{g=n_i+1}^{n+1} R_g) = \frac{1}{1-\rho}\sum_{g=1}^{n_i} R_g. \tag{8}$$

Note that $ld(C_i) > ad(C_i)$ can occur when $n_i \geq (n-f)/2$ and thus absorption is not ensured in all cases that $dist(C_i, C_j) \leq ld(C_i)$ *prior* to fire of the first node in C_i. Hence we say that if $dist(C_i, C_j) \leq ld(C_i)$ then the clusters are *prone* to absorption.

Step 2: We claim that if the following relationship between ad, ld and $cycle$, is satisfied, then absorption is ensured irrespective of the states of the clusters. WLOG let C_c be the largest cluster.

$$\sum_{j=1}^{c-1} ad(C_j) + \overline{ld(C_c)} \geq cycle. \tag{9}$$

The following theorem shows the requirement for the continuous existence of at least two synchronized clusters that undergo absorption of each other.

Theorem 2 *For n, f, cycle and a given clustering of $n - f$ correct nodes into $c > 1$ clusters at time t_0, for which Eq. 9 is satisfied, there will be at least one cluster that will absorb some other cluster by time $t_0 + (2\,cycle)$.*

Proof: We assume WLOG that C_j is the largest cluster when running the clustering procedure at some time t_0, at which some node fires. Let $t_1 \geq t_0$ denote the time at which the last node in C_1 fires.

Assume that k is the number of inputs from nodes that are not in C_1 that caused a node in C_1 to fire (not counting the fire from the cluster itself). If $k > f$ then at least one correct node caused the nodes to fire. Theorem 1 implies that C_1 is absorbing with the cluster of that node and therefore the objective is reached. We therefore assume $k \leq f$ and that by firing C_1 did not absorb with any preceding cluster.

If any other cluster fires between t_0 and t_1 then it has absorbed with C_1 and we are done.

Thus, assume that no other cluster besides C_1 fires between t_0 and t_1. Therefore, we can still refer to the same cluster partition at t_1, without the need to run the clustering procedure again. Exactly one of the following two possibilities should take place at t_1:

1. $\exists i$ $(1 \leq i \leq c)$, such that $dist(C_i, C_{(i+1)\,(\mathrm{mod}\ c)}) \leq ad(C_i)$.
2. $\forall i$ $(1 \leq i < c, i \neq j), dist(C_i, C_{i+1}) > ad(C_i)$ and due to Eq. 9, $dist(C_j, C_{(j+1)\,(\mathrm{mod}\ c)}) \leq ld(C_j)$.

Assume the first possibility, and further assume that all clusters up to C_{i-1} fire successively without causing the absorption of C_i or of any other cluster. Thus, we can still refer to the original clustering, by the time C_i fires. C_i then necessarily fires due to a $0 \leq k \leq f$ sized input following the assumption that it did not absorb with a preceding cluster. Following Theorem 1, C_i absorbs C_{i+1}.

Assume the second possibility, and further assume that there is no absorption until C_j fires. Let t_j denote the time at which the last node in C_j fires. There are two possibilities at t_j:

a. $\exists i(1 \leq i < c)$, such that at $t_j, dist(C_i, C_{i+1}) \leq ad(C_i)$.

b. $\forall i(1 \leq i < c, i \neq j), dist(C_i, C_{i+1}) > ad(C_i)$ and therefore due to Eq. 9, $dist(C_j, C_{(j+1)\,(\mathrm{mod}\ c)}) \leq ld(C_j)$.

Assume possibility a. This means that at time t_j, it could be that $dist(C_j, C_{(j+1)\,(\mathrm{mod}\ c)}) > ld(C_j)$ and therefore no absorption of C_j and $C_{(j+1)\,(\mathrm{mod}\ c)}$ following the fire of C_j . On the other hand it reduces the given clustering into possibility (1.) above, in which it was proven that absorption is ensured at the next cycle, by the time cluster C_i begins its next round of firing.

Assume possibility b. Because $dist(C_j, C_{(j+1)\,(\mathrm{mod}\ c)}) \leq ld(C_j)$ at t_j there will be an absorption of C_j and $C_{(j+1)\,(\mathrm{mod}\ c)}$ following the fire of C_j , and thus the proof is concluded.

□

The following theorem assumes the worst case of $n = 3f + 1$. In Section 5 we discuss various ratios between them and also about the relationship between the cycle length and f.

Theorem 3 *Within at most $2(2f + 1)$ cycle time units the pulse synchronization algorithm reaches a synchronized state of the system.*

Proof: Assume that $n = 3f + 1$. Thus, the maximal number of clusters is $2f + 1$, and since at least two clusters absorb in every two cycles we obtain the bound. □

4.3 A solution to the set of constraints

Incorporating the absorption distance of Theorem 1 and Eq. 8 into Eq. 9 yields,

$$\sum_{j=1}^{c-1}[\frac{1}{1-\rho}\sum_{i=f+1}^{f+n_j}R_i - \frac{2\rho}{1-\rho^2}\sum_{i=f+1}^{n+1}R_i] + \frac{1}{1-\rho}\sum_{i=1}^{n_c}R_i \geq cycle \Rightarrow$$

$$\sum_{j=1}^{c-1}[\sum_{i=f+1}^{f+n_j}R_i - \frac{2\rho}{1+\rho}\sum_{i=f+1}^{n+1}R_i] + \sum_{i=1}^{n_c}R_i \geq (1-\rho)\, cycle \Rightarrow$$

$$\sum_{j=1}^{c-1}\sum_{i=f+1}^{f+n_j}R_i + \sum_{i=1}^{n_c}R_i \geq (1-\rho)\, cycle + (c-1)\,\frac{2\rho}{1+\rho}\sum_{i=f+1}^{n+1}R_i. \tag{10}$$

$$(1-\rho)\, cycle + (c-1)\,\frac{2\rho}{1+\rho}\sum_{i=f+1}^{n+1}R_i \leq$$

$$(1-\rho)\, cycle + (c-1)\,\frac{2\rho}{1+\rho}\sum_{i=1}^{n+1}R_i < (1+\frac{2\rho}{1+\rho}(n-f))\, cycle. \tag{11}$$

Using Eq. 10 and Eq. 11 the constraint takes the following structure:

$$\sum_{j=1}^{c-1}\sum_{i=f+1}^{f+n_j}R_i + \sum_{i=1}^{n_c}R_i \geq (1+\frac{2\rho}{1+\rho}(n-f))\, cycle, \text{where } \sum_{j=1}^{c}n_j = n-f. \tag{12}$$

Eq. 12 specifies a linear equation on the variables R_i. A solution with respect to the boundary case of a **specific** partitioning of exactly $n-f$ correct nodes into separate clusters ensures absorption. Any R that satisfies all linear equations, whose set reflect all possible partitions into clusters, ensures continuous absorption until a synchronized state of the system is reached.

Corollary 4.6 *For a given cycle, f and n, finding a solution to the problem of "pulse synchronization" employing the proposed algorithm, reduces to solving the set of linear constraints on R determined by Eq. 12.*

Solving Eq. 12, under Restrictions 2-5, for some typical parameters and several small n's allowed the guessing of a solution (see Section A.1) that is proven in Theorem 4 to hold for every n and $f < n/3$.

Theorem 4 *Given[2] $n > 3f+1$ and cycle, the refractory function:*

$$R_i = \begin{cases} \frac{(1+\frac{2\rho}{1+\rho}(n-f))\, cycle}{n-f} & i = 1\ldots n-f-1 \\ \frac{R_1 - R_{n+1} - \frac{2\rho}{1+\rho}(n-f)\, cycle}{f+1} & i = n-f\ldots n \\ 2d(1+\rho)\frac{(\frac{1+\rho}{1-\rho})^{n+3}-1}{(\frac{1+\rho}{1-\rho})-1} & i = n+1, \end{cases} \tag{13}$$

constitutes a solution to the set of linear equations of R.

[2]It can be shown that only the first $\max[f+\lfloor(n-f)/2\rfloor+1, n-f]$ variables participate in the constraints. For the case $n \geq 3f+2$, we get that $n-f-1 \geq f+\lfloor(n-f)/2\rfloor$. Theorem 4 presents a solution for this case. For the case of $n = 3f+1$ we get that $n-f-1 < f+\lfloor(n-f)/2\rfloor$. Thus, $f+\lfloor(n-f)/2\rfloor = (4n-1)/6$, replaces the bound of $n-f-1$ in Restriction 3 and consequently also in Theorem 4. An equivalent proof to Theorem 4 can be given for the case $n = 3f+1$, Theorem 6 (Appendix, Section A.2 presents a solution for the latter case

Proof: The steps of the proof:

1. Prove that the solution renders all the constraints implied by Eq. 12 equally restrictive and show that they are satisfied by the proposed solution of Eq. 13.

2. Prove that $R_1 + \cdots + R_{n+1} = cycle$.

Step 1. Incorporating the solution Eq. 13 into Eq. 12 implies that all the variables with non-zero coefficients $R_1 \ldots R_{n-f-1}$ are equal. This reduces the set of all possible equations determined by Eq. 12 to a single equation: $(n-f)R_i \geq (1 + \frac{2\rho}{1+\rho}(n-f))\, cycle$, for any i, $1 \leq i \leq n-f-1$. Thus, Eq. 13 satisfies this constraint: $(n-f)R_i = (n-f)\frac{(1+\frac{2\rho}{1+\rho}(n-f))\, cycle}{n-f} = (1 + \frac{2\rho}{1+\rho}(n-f))\, cycle$.

Step 2.

$$\begin{aligned}
&R_1 + \cdots + R_{n+1} = (R_1 + \cdots + R_{n-f-1}) + (R_{n-f} + \cdots + R_n) + R_{n+1} = \\
&\quad (n-f-1)\frac{(1 + \frac{2\rho}{1+\rho}(n-f))\, cycle}{n-f} + (f+1)\frac{R_1 - R_{n+1} - \frac{2\rho}{1+\rho}(n-f)\, cycle}{f+1} + R_{n+1} = \\
&\quad (1 + \frac{2\rho}{1+\rho}(n-f))\, cycle - \frac{(1 + \frac{2\rho}{1+\rho}(n-f))\, cycle}{n-f} + R_1 - R_{n+1} - \frac{2\rho}{1+\rho}(n-f)\, cycle + R_{n+1} = \\
&\quad (1 + \frac{2\rho}{1+\rho}(n-f))\, cycle - \frac{2\rho}{1+\rho}(n-f)\, cycle = cycle.
\end{aligned}$$

Therefore, Eq. 13 comprises a solution to the set of constraints for the case of $0 < 3f + 1 < n$. This solution complies with Restrictions 2-5 for a certain range of f and $cycle$ (cf. Eq. 14). □

Theorem 5 *The algorithm of Section 3 solves the Pulse Synchronization Problem.*

Proof: Theorem 4 proves that Eq. 13 solves the set of linear constraints on R, (Eq. 12) for a given $cycle$, f and n, and any possible partitioning of $n-f$ correct nodes into synchronized clusters. Corollary 4.6 thus implies that employing the algorithm of Section 3 with the refractory function R of Eq. 13 comprises a solution to the pulse synchronization problem. □

5 Analysis of the algorithm and comparison to related algorithms

Our protocols do not depend explicitly on the actual behavior of the nodes. The convergence of the clusters depends on having at most f nodes that may behave arbitrarily. Therefore, our protocols can withstand any faulty behavior, including dynamic failure and recovery of nodes (intermittent faults). Nodes that have just recovered need time to synchronize, therefore, we assume that eventually we have a window of time within which there are no newly recovered nodes, and within which the system inevitably converges (Theorem 2).

Fault ratio: Many fundamental problems in distributed networks have been proven to require $3f + 1$ nodes to overcome f concurrent faults in order to reach a deterministic solution without authentication [11, 13, 5, 4]. We have not shown this ratio to be a necessary requirement for solving the “pulse synchronization” problem but the results for related problems lead us to believe that a similar result may exist for the “pulse synchronization” problem.

Time complexity: A randomized self-stabilizing Byzantine digital clock synchronization algorithm in [7, 8] designed for a fully connected communication graph synchronizes in $M2^{2(n-f)}$ pulses, where M is the upper bound on the clock values held by individual processors. The algorithm uses broadcast message passing, it allows transient and permanent faults during convergence, requires at least $3f+1$ processors, but utilizes a global pulse system. An additional algorithm in [8], does not use a global pulse system and is thus partially synchronous similar to our model. The convergence time of the latter algorithm is $O((n-f)n^{6(n-f)})$. This is drastically higher than our result even if one takes into consideration the relationship between *cycle* and f of our solution(Eq. 14), which yields an $O(f^3)$ time units convergence complexity.

Our use of broadcasts is essentially similar to the type of broadcast used in [8], since we assume nothing on the order of arrival of the messages and concomitant messages from the same source are ignored. Thus, a faulty node can send two consecutive messages with differing values.

Message and space complexity: Each correct node multicasts exactly one message per cycle. This yields a message complexity of at most n messages per cycle. The system's message complexity to reach synchronization from any arbitrary state is at most $2n(2f+1)$ messages per synchronization from any arbitrary initial state. The faulty nodes cannot cause the correct nodes to fire more messages during a cycle. Comparatively, the self-stabilizing clock synchronization algorithm in [8] sends n messages during a pulse and thus has a message complexity of $O(n(n-f)n^{6(n-f)})$. This is significantly larger than our message complexity irrespective of the time interval between the pulses.

The space complexity is $O(n)$ since the variables maintained by the processors keep only a linear number of messages recently received and various other small range variables. The number of possible states of a node is linear in n, but the node does not need to keep a configuration table.

Tightness of synchronization: Using the presented algorithm, the invocation of the pulses of the nodes will be synchronized to within d real time units. This asymptotically equals the lower bound on clock synchronization $d(1-1/n)$ in completely connected, fault-free networks [12]. Were our algorithm to be used as a logical clock synchronization algorithm then the clocks would show a maximum time difference of $d(1+\rho)$ between each other due to the hardware clock skew. Comparatively, the clock synchronization algorithm of [5] reaches a synchronization tightness of $d(1+\rho)+2\rho(1+\rho)R$ where R is the time between synchronizations. The second Byzantine digital clock synchronization algorithm in [8] reaches a synchronization tightness which is in the magnitude of $(n-f)d(1+\rho)$. This is significantly less tight than our result.

Firing frequency bound ("Linear Envelope"): The firing frequency bound is around twice that of the original firing frequency of the nodes (linear envelope). This bound is determined by the fault ratio (the sum of the first f threshold steps relative to the cycle length). For $n=3f+1$ it translates to $\approx \frac{1}{2}cycle$. Thus, if required, the firing frequency bound can be closer to the endogenous firing frequency of 1 *cycle* if the fraction of faulty nodes is assumed to be lower. For example, for a fraction of $n=10f$, the lower bound on the cycle length is approximately 8/9 that of the endogenous cycle length.

The allowed parameter range (of n and *cycle*): Restrictions 2-5 determine the lower bound on the allowed length of each threshold step, R_i. There are exactly $n+1$ threshold steps whose sum needs to equal exactly *cycle* time units. Restrictions 2-5 therefore also determine a required relationship between the input parameters n and *cycle*.

To show the relationship between the input parameters n, f and *cycle* and that the solution space is not empty we need to verify that given n, f and *cycle* then the solution (Eq. 13) complies with:

1. R_i, $i = 1 \ldots n-f-1$, comply with Restrictions 3 and 4:
$$\frac{(1+\frac{2\rho}{1+\rho}(n-f))\, cycle}{n-f} > \max(3d, d(1+\rho) + \frac{\rho}{1+\rho}\, cycle).$$

2. R_i, $i = n-f \ldots n$, comply with Restriction 4:
$$\frac{R_1 - R_{n+1} - \frac{2\rho}{1+\rho}(n-f)\, cycle}{f+1} > d(1+\rho) + \frac{\rho}{1+\rho} \sum_{j=n-f}^{n+1} R_j.$$

3. R_i, $i = 1 \ldots n-f-1$ comply with: $R_i > \sum_{j=n-f}^{n+1} R_j$.

This yields the following approximate required relationship between f and *cycle* (assuming $n = 3f+1$), ensuring that the solution of (Eq. 13) complies with Restrictions 2-5:

$$4df^2 < cycle. \tag{14}$$

A disadvantage of the algorithm is thus its scalability with respect to the fault ratio and cycle length should the fault ratio be taken at its bound, $n = 3f+1$. This relationship improves though, with a decrease in d, the network delay. It can be shown that a fault ratio of $n \leq 3f$ introduces a contradiction into the set of equations governed by Eq. 12. This is the source of the requirement $n \geq 3f+1$. If we fix the solution for a boundary network size of $3f+1$ and designate it as m, then any network size of $n \geq m$ is acceptable.

6 Discussion

We developed a self-stabilizing fault tolerant algorithm for *pulse synchronization* in distributed computer networks. The pulse synchronization problem resembles the clock synchronization problem. The linear envelope requirement defines a restriction that defies a straight forward simple solution. Moreover, the fact that there isn't a specific value that describes the pulse difference among nodes, eliminates the ability to use some sort of approximate approach ([6]) or consensus. Clock synchronization can be viewed as approximate agreement on a value representing real time.

The algorithm developed is inspired by and shares properties with the lobster cardiac pace-maker network; the network elements fire in tight synchrony within each other, whereas the synchronized firing pace can vary, up to a certain extent, within a linear envelope of a completely regular firing pattern. This behavior can be compared to the requirement of keeping the clocks of the network computers synchronized with each other ("precision") as well as being close to real time ("accuracy"). Pulse synchronization and clock synchronization have similar precision requirements but differ in the accuracy requirement. Algorithms that guarantee a high level of accuracy also guarantee a high level of precision, but the opposite is not implied. We suggest that when a high level of precision is required but the accuracy requirement is not high, a simpler algorithm can be employed. A pulse synchronization algorithm may replace a clock synchronization algorithm in such cases.

Possible applications: We offer the Byzantine self-stabilizing pulse synchronization scheme to drastically improve convergence time, scalability and synchronization tightness when there is no real need to actually agree on the pulse counters, as in digital clock synchronization.

A self-stabilizing pulse synchronization algorithm can be beneficial to solve TDMA based slot synchronization (in multi access channels such as cellular phones; slotted ALOHA) [19]. A similar suggestion appears in [14], though without supplying any fault tolerance. In this problem, time is divided into frames of fixed length, and each frame contains a certain number of slots. Frames are repeated periodically, and each participant is exclusively allocated one slot per frame. It is of paramount importance to avoid collisions of the slot allocations. If the participants are not tightly synchronized with respect to the frame starting point then collisions will occur. The exact starting time of the frame is of minor importance relative to the synchronization requirement. Should a central synchronizer be present then the network would be extremely sensitive to faults in the central station. Where no central station is available and connectivity changes rapidly a self-stabilizing distributed synchronization protocol would make the participants resynchronize without any exterior intervention, should the participant be out of synchronization. Other cases that could benefit from a self-stabilizing pulse synchronization scheme include global snapshots; load balancing; distributed inventory count; distributed scheduling; backup of distributed systems; debugging of distributed programs; and deadlock detection.

Possible biological implications: We may now speculate on possible new meanings of the refractory function in the biological context. Could there be some correlation between the number of faults and their severity that the biological neuronal network must face and the network size required to reach synchronization? Could there be some relationship between the shape of the neuronal refractory function and the size of the biological neural network? Does the shape of the refractory function of a single neuron play a role in the networks ability to tolerate faults or noise? Does the shape of the refractory function of a single neuron play a role in the networks ability to alter the synchronized firing pace?

References

[1] G. E. Andrews, "*The Theory of Partitions*", **Encyclopedia of Mathematics and Its Applications**, Vol. 2, Addison-Wesley, Reading, MA, 1976.

[2] J. Brzeziǹski, and M. Szychowiak, "*Self-Stabilization in Distributed Systems - a Short Survey*, **Foundations of Computing and Decision Sciences**, Vol. 25, no. 1, 2000.

[3] J. Buck, and E. Buck, "*Synchronous fireflies*", **Scientific American**, Vol. 234, pp. 74-85, May 1976.

[4] D. Dolev, J. Halpern, and H. R. Strong, "*On the Possibility and Impossibility of Achieving Clock Synchronization*", **J. of Computer and Systems Science**, Vol. 32:2, pp. 230-250, 1986.

[5] D. Dolev, J. Y. Halpern, B. Simons, and R. Strong, "*Dynamic Fault-Tolerant Clock Synchronization*", **J. Assoc. Computing Machinery**, Vol. 42, No.1, pp. 143-185, Jan. 1995.

[6] D. Dolev, N. A. Lynch, E. Stark, W. E. Weihl and S. Pinter, "*Reaching Approximate Agreement in the Presence of Faults*, **Journal of the ACM**, 33 (1986) 499-516.

[7] S. Dolev, "*Self-Stabilization*," **The MIT Press**, 2000.

[8] S. Dolev, and J. L. Welch, "*Self-Stabilizing Clock Synchronization in the presence of Byzantine faults*", **Proc. Of the Second Workshop on Self-Stabilizing Systems**, pp. 9.1-9.12, 1995.

[9] WO. Friesen, "*Physiological anatomy and burst pattern in the cardiac ganglion of the spiny lobster Panulirus interuptus*", **J. Comp. Physiol.**, Vol. 101, pp. 173-189, 1975.

[10] WO. Friesen, "*Synaptic interaction in the cardiac ganglion of the spiny lobster Panulirus interuptus*", **J. Comp. Physiol.**, Vol. 101, pp. 191-205, 1975.

[11] M. J. Fischer, N. A. Lynch and M. Merritt, "*Easy impossibility proofs for distributed consensus problems*", **Distributed Computing**, Vol. 1, pp. 26-39, 1986.

[12] J. Lundelius, and N. Lynch, "*An Upper and Lower Bound for Clock Synchronization*," **Information and Control**, Vol. 62, pp. 190-205, Aug/Sep. 1984.

[13] N. Lynch, "*Distributed Algorithms*", **Morgan Kaufmann**, 1996.

[14] R. Mathar and J. Mattfeldt, "*Pulse-coupled decentral synchronization*", **SIAM J. Appl. Math**, Vol. 56, No. 4, pp. 1094-1106, Aug. 1996.

[15] R. E. Mirollo and S. H. Strogatz, "*Synchronization of pulse-coupled biological oscillators*", **SIAM J. Appl. Math**, Vol. 50, pp. 1645-1662, 1990.

[16] Z. Nèda, E. Ravasz, Y. Brechet, T. Vicsek, and A.-L. Barabàsi, "*Self-organizing process: The sound of many hands clapping*", **Nature**, 403, pp. 849-850, 2000.

[17] E. Sivan, H. Parnas and D. Dolev, "*Fault tolerance in the cardiac ganglion of the lobster*", **Biol. Cybern.**, Vol. 81, pp. 11-23, 1999.

[18] S. H. Strogatz and I. Stewart, "*Coupled Oscillators and Biological Synchronization*", **Scientific American**, Vol. 269, pp. 102-109, Dec. 1993

[19] A. S. Tanenbaum, "*Computer Networks* 3^{rd} *ed.*", **Prentice Hall International**, 1996.

[20] T. J. Walker, "*Acoustic synchrony: two mechanisms in the snowy tree cricket*", **Science**, 166, pp. 891-894, 1969.

A Appendix

A.1 The problems of solving the set of constraints of Eq. 12 analytically

As mentioned in Section 4.3 each constraint is determined by a partitioning of the correct nodes into disjoint clusters. The calculation of the possible number of such partitions equals the problem of integer partitioning, i.e. the number of ways of writing the integer k as a sum of positive integers irrespective of the order, denoted $P(k)$ in the literature. The following asymptotic result to the estimation of $P(k)$ is given in [1]:

$$P(k) \sim \frac{1}{4k\sqrt{3}} e^{\pi\sqrt{2k/3}} \tag{15}$$

Some numerical examples derived of Eq. 15:

$$\begin{array}{lcl} P(4) & = & 5. \\ P(50) & = & 204226. \\ P(1000) & = & 24061467864032622473692149727991. \end{array}$$

Clearly solving this set of linear constraints analytically or conduct Gaussian elimination is unfeasible even for small ns. Therefore, another method for reaching a solution must be found. Hence, we guessed the solution presented in Theorem 4.

A.2 Proof of Theorem 4 for $n = 3f + 1$

The ld, which is associated with the largest cluster, contributes a larger number of R_i steps than any individual ad does. Note that the specific partitioning into clusters determines which R_i have non zero coefficients in a specific constraint. It follows, that in order to generalize this property we need to assess what clustering yields the highest index of R_i participating in the constraints and thus we can conclude what variables always have zero coefficients in all the constraints. The largest possible cluster in an unsynchronized state is exposed in the scenario of two clusters in which, WLOG, $n_1 = 1$ and $n_2 = n - f - 1$, yielding $ld(C_c) = R_1 + \cdots + R_{n-f-1}$. The second largest cluster that can exist (i.e. largest cluster represented by an ad in Eq. 9), is when there are two equally or almost equally sized clusters, $n_1 = \lceil (n-f)/2 \rceil$ and $n_2 = \lfloor (n-f)/2 \rfloor$. This implies that an individual ad equals at most $R_{f+1} + \cdots + R_{f+\lfloor (n-f)/2 \rfloor}$. The conclusion is that the variables $R_{\max(f+\lfloor (n-f)/2 \rfloor +1, n-f)} \ldots R_{n+1}$ always have a zero coefficient in any constraint determined by Eq. 12. This implies that if $n - f \geq f + \lfloor \frac{n-f}{2} \rfloor$ then:

There are at most

$$n - f - 1 \text{ variables with a non-zero coefficient.} \tag{16}$$

On the other hand there are $n - f$ correct nodes, each one contributing exactly one added value of "1" to any constraint derived from Eq. 12, therefore:

$$\text{The sum of all the coefficients in any constraint equals exactly: } n - f. \tag{17}$$

Let us observe the scenario of two clusters, C_1 and C_2, such that the cardinality of the largest cluster of the two, WLOG C_2, $n_2 = f$. This scenario yields the constraint $(R_1 + \cdots + R_f) + (R_{f+1} + \cdots + Rf + n_1) \geq cycle$ which we will rewrite as

$$R_1 + \cdots + R_{f+n_1} \geq R_1 + \cdots + R_{n+1}. \tag{18}$$

$f + n_1 < n + 1$, since $f + n_1 + n_2 = n$. Therefore due to the non-negativity of R_i Eq. 18 is a contradiction. The same arguments and conclusions hold for any $n_2 \leq f$ and its respective constraint.

To overcome this problem we need the additional constraint, $n_2 > f$, on the largest cluster. The largest possible cluster, n_2, reaches a minimum for $n_1 = n_2$ (otherwise n_1 would have represented the largest of the two clusters). This additional constraint is equivalent to $2n_2 > 2f \Rightarrow n - f > f$ which gives the relationship:

$$n > 3f. \tag{19}$$

Proof of Theorem 4 for $n = 3f + 1$
$\Rightarrow f + \lfloor \frac{n-f}{2} \rfloor = \max\left(f + \lfloor \frac{n-f}{2} \rfloor, n - f - 1\right)$:

The adjusted constraint:

Restriction 3.1: $R_i > 3d$, $i = 1 \dots \frac{4n-1}{6}$.

Theorem 6 *Given $n = 3f + 1$, and cycle, the refractory function:*

$$R_i = \begin{cases} \frac{6}{4n+5}(1 + \frac{2\rho}{1+\rho}(n-f))\,cycle & i = 1 \dots \frac{4n-1}{6} \\ \frac{6R_1 - R_{n+1} - \frac{2\rho}{1+\rho}(n-f)\,cycle}{\frac{2n+1}{6}} & i = \frac{4n-1}{6} + 1 \dots n \\ 2d(1+\rho)\frac{(\frac{1+\rho}{1-\rho})^{n+3} - 1}{(\frac{1+\rho}{1-\rho}) - 1} & i = n + 1, \end{cases} \tag{20}$$

constitutes a solution to the set of linear equations of R.

Proof: The steps of the proof:

1. Prove that the solution renders all the constraints implied by Eq. 12 equally restrictive and show that they are satisfied by the proposed solution of Eq. 20
2. Prove that $R_1 + \cdots + Rn + 1 = cycle$.

Step 1. Incorporating the solution, Eq. 20, into Eq. 12 implies that all the variables with non-zero coefficients $R_1 \dots R_{(4n-1)/6}$ are equal. This reduces the set of all possible equations determined by Eq. 12 to a single equation: $\left(\frac{4n-1}{6} + 1\right) R_i \geq (1 + \frac{2\rho}{1+\rho}(n-f))\,cycle$, for i, $1 \leq i \leq \frac{4n-1}{6}$. Thus, Eq. 20 satisfies:

$$\left(\frac{4n-1}{6} + 1\right) R_i = \frac{4n+5}{6}\,\frac{6(1-\rho)\,cycle}{4n+5} = (1 + \frac{2\rho}{1+\rho}(n-f))\,cycle.$$

Step 2.

$$\begin{aligned} &R_1 + \cdots + R_{n+1} = (R_1 + \cdots + R_{(4n-1)/6}) + (R_{(4n-1)/6)+1} + \cdots + R_n) + R_{n+1} = \\ &\quad \frac{4n-1}{6}\,6(1 + \frac{2\rho}{1+\rho}(n-f))\,cycle + \frac{2n+1}{6}\,\frac{6R_1 - R_{n+1} + \rho\,cycle}{\frac{2n+1}{6}} + R_{n+1} = \\ &\quad \frac{4n+5-6}{6}\,\frac{6}{4n+5}(1 + \frac{2\rho}{1+\rho}(n-f))\,cycle + 6R_1 - R_{n+1} - \frac{2\rho}{1+\rho}(n-f)\,cycle + R_{n+1} = \\ &\quad (1 + \frac{2\rho}{1+\rho}(n-f))\,cycle - 6\left(\frac{6}{4n+5}(1 + \frac{2\rho}{1+\rho}(n-f))\,cycle\right) + \\ &\qquad\qquad +6R_1 - R_{n+1} - \frac{2\rho}{1+\rho}(n-f)\,cycle + R_{n+1} = \\ &\quad (1 + \frac{2\rho}{1+\rho}(n-f))\,cycle - \frac{2\rho}{1+\rho}(n-f)\,cycle = cycle. \end{aligned}$$

Therefore, Eq. 20 comprises a solution to the set of constraint for the case of $n = 3f + 1$. This solution complies with Restrictions 2, 3.1, 4 and 5 for a certain range of f and *cycle* (cf. Eq. 14). □

A.3 The Summation algorithm

Objective:
See Restriction 1.
Heuristics:

1. When the input counter crosses the threshold level, either due to a sufficient counter increment or a threshold decrement, then the node signs and sends a message (fires). The message sent holds the value of Counter at sending time.

2. "timeliness procedure" is employed at the receiving node to assess the credibility (timeliness) of the value of the Counter contained in this message. The timeliness procedure ensures that messages sent by correct nodes with Counter less than n will always be assessed as timely by other correct receiving nodes.

3. When a received message is declared timely and therefore accounted for it is stored in a "counted" message buffer ("Counted State"). The receiving node's Counter is then updated to hold a value greater than the Counter in the message.

4. If a message received is declared untimely then it is temporarily stored in an "uncounted" message buffer ("Uncounted State") and will not be accounted for at this stage. Over time, the timeliness test of previously stored timely messages may not hold any more. In this case, such messages will be moved from the Counted State to the Uncounted State.

5. All messages are deleted after a certain time-period.

Definitions and state variables:
Counter: an integer representing the node's estimation of the number of timely firing events received from distinct nodes within a certain time window. Counter is updated upon receiving a timely message. The node's Counter is checked against the refractory function whenever one of them changes. The value of Counter is bounded and non-monotonous; the arrival of timely events may increase it and the decay/untimeliness of old events may decrease it.
Signature entry: a basic data structure represented as (S_p, t_{arr}) and created upon arrival of a message M_p. S_p is the *id* (or signature) of the sending node p and t_{arr} is the arrival time of the message. We say that two signature entries, (S_p, t_1) and (S_q, t_2), are **distinct** if $p \neq q$.
Counted State (CS): a set of distinct signature entries that determine the current value of Counter. The Counter reflects the number of signature entries in the Counted State. A signature entry is **accounted for** in Counter, if it was in CS when the current value of Counter was determined. We require exclusive write access to CS.
Uncounted State (UCS): a set of signature entries, not necessarily distinct, that have not been accounted for in the current value of Counter and that are not yet due to decay. A signature entry is placed in the UCS when its message clearly reflects a faulty sending node or because it is not timely anymore.

Retired UCS (RUCS): a set of distinct signature entries not accounted for in the current value of Counter due to the elapsed time since their arrival. These signature entries are awaiting deletion (decaying).

The CS and UCS are mutually exclusive and together reflect the relevant messages received from other nodes in the preceding time window. Their union is the node's **message state.**

We use the notation $Counter_q$ to mark node q's Counter, $Counter_{M_p}$ to mark the Counter contained in a message M_p sent by p and |message state| to mark the number of distinct signature entries in the message state.

$T(p, M_p)$: denotes the time at node p at which it sent the message M_p.

$T(M_p, q)$: denotes the arrival time of the message M_p at node q.

$MessageAge(t, p)$: the elapsed time on a node's clock since the most recent arrival of a message from node p. Thus, its value at a node q at time t, is given by $t - T(M_p, q)$.

$StateAge(t, k)$: denotes $MessageAge(t, \ldots)$ of the k'th most recent signature entry in the node's message state.

$CSAge(t)$: denotes, at t, the largest $MessageAge(t, \ldots)$ among the signature entries in CS.

Procedures used by the Summation algorithm:

We say that M_p has been **assessed** by q, once the following procedure is completed. A message M_p, is timely at time t_{arr} at node q once it is declared timely by the procedure.

TIMELINESS (M_p, t_{arr}):

Timeliness Condition 1:

If ($0 > Counter_{M_p}$ or $Counter_{M_p} > n - 1$) then
  return "M_p is not timely";

Timeliness Condition 2:

Create a new signature entry (S_p, t_{arr}) and insert it into UCS;
  If ($\exists(S_p, t)$ s.t. $t \neq t_{arr})$[3] in message state $\cup RUCS$) then
    delete from message state all (S_p, t'), where $t' \neq \max(T(M_p,q))$;
    return "M_p is not timely";

Timeliness Condition 3:

Let k denote $Counter_{M_p}$, and let $\tau(k) = 2d(1+\rho)\frac{(\frac{1+\rho}{1-\rho})^{k+1}-1}{(\frac{1+\rho}{1-\rho})-1}$.
  If ($k <$ |message state|) AND ($StateAge(t_{arr}, k+1) \leq \tau(k)$)
  at some time in the interval $[t_{arr}, t_{arr} + d(1+\rho)]$[4] then
    return "M_p is timely";
  else
    return "M_p is not timely";

The following procedure atomically moves and deletes obsolete signature entries. It prunes the CS to hold only signature entries such that a message sent holding the resultant Counter will be assessed as timely at any correct receiving node. Counter is then updated.

PRUNE(t):

- Delete (decay) from RUCS all signature entries (S_p, t) whose $MessageAge(t, p) > \tau(n+2)$;

[3] We assume no concomitant messages are stamped with the exact same arrival times at a correct node. We assume that one can uniquely identify messages.

[4] We assume the implementation detects that these conditions are satisfied within the time window.

- Move to RUCS, from the message state, all signature entries (S_p, t) whose $MessageAge(t, p) > \tau(n+1)$;
- Move to UCS, from CS, signature entries, beginning with the oldest, until: $CSAge(t) \leq \tau(k-1)$, where $k = \max(1, |CS|)$;
- Set $Counter = |CS|$;

The following procedure atomically moves signature entries from UCS into CS and updates the value of Counter following the arrival of a new timely message M_p.

MAKE-ACCOUNTABLE(M_p):

- Move the $\max(1, Counter_{M_p} - Counter_q + 1)$ most recent distinct signature from UCS to CS;
- Set $Counter = |CS|$;

The following procedure generates the signature S_q, of a node q.

SIGN: $S_q \equiv \text{Sign}(Counter_q)$[5];

SEND: Multicast S_q.

The set of procedures given above comprise the elements of the event driven Summation Algorithm.

Lemma A.1 *The Counter of a correct node cannot exceed n and a correct node will not send a Counter that exceeds $n-1$.*

Proof: There can be at most n distinct signature entries in the CS of a correct node therefore bounding the Counter by n. A Counter that equals exactly n implies the node's own signature entry is in the CS. If the node assesses it's own message $d(1+\rho)$ time units after sending it then it's own signature entry will decay exactly $\tau(n+2) + d(1+\rho)(< \tau(n+3))$ time units after it was sent. Consequently, the Counter will decrease and will be at most $n-1$. The node will still be in R_{n+1} subsequent to this decay implying the node can fire only when the Counter is at most $n-1$. □

Let $t_{\text{send}M_q}$ denote the time at which a receiving node q is ready to assess whether to send a message following the arrival and processing of a message M_p. Denote by $t_{\text{send}M_p}$ the time at which p sent M_p.

Lemma A.2 *A signature entry, which has been moved to the RUCS of a correct node q prior to t_{sendM_p}, cannot have been accounted for in $Counter_{M_p}$.*

Proof: Assume that the signature entry (S_r, t') was moved to RUCS of node q at a time t that is smaller than $t_{\text{send}M_p}$. Thus at q at time t, $MessageAge(t, r) > \tau(n+1)$. Therefore at node p at $t_{\text{send}M_p}$, $MessageAge(t_{\text{send}M_p}, r) > \tau(n)$.

Following the PRUNE procedure at p, $(S_r, t")$ would have been accounted for at the sending time of M_p only if $Counter_{M_p} \geq n+1$. Therefore by Lemma A.1 node p did not account for the signature entry of r in $Counter_{M_p}$. □

[5] With authenricated channels there is no actual need to encrypt the signature/message. The signatures in the CS are never authenticated beyond arrival, making the need to hold a digital signature in the signature entry redundant, the node id is enough.

Lemma A.3 *A signature entry that has decayed at a correct node q prior to* t_{sendM_p}*, can not have been accounted for in* $Counter_{M_p}$.

Proof: Lemma A.3 is an immediate corollary of Lemma A.2. □

Lemma A.4 *Let* **age** *denote* $MessageAge(t_{prune}, \ldots)$ *of the oldest signature entry, at the time of execution of the* PRUNE *procedure at a correct node p, accounted for in its timely message* M_p *with* $Counter_{M_p}$*. Then*

$$StateAge(t_{send}, Counter_{M_p} + 1) \leq age\frac{1+\rho}{1-\rho} + 2d(1+\rho).$$

Proof: The $MessageAge(t_{\text{prune}}, \ldots)$ of the oldest signature entry accounted for in $Counter_{M_p}$ is given by age at p at time t_{prune}, the time it executed the PRUNE procedure. The message of this oldest signature entry could have arrived at q, $\delta(1+\rho)$ time units on q's clock, prior to its arrival at p, by which time p would also have received all the messages that this message has accounted for. Another $\pi(1+\rho)$ time units could then have passed on q's clock until M_p was sent. M_p could have arrived at q, $\delta(1+\rho)$ time units on q's clock after it was sent by p. By this time q would also have received all the messages that are accounted for in M_p. Another $\pi(1+\rho)$ time units can then pass on q's clock until $t_{\text{send}M_p}$. Any additional messages arriving at q until time $t_{\text{send}M_p}$, can only decrease $StateAge(t_{\text{send}M_p}, Counter_{M_p}+1)$ since the $(k+1)^{\text{th}}$ most recent signature entry at q will then be more recent than the oldest signature entry accounted for in $Counter_{M_p}$. More specifically, at q's prune time t_{prune} and at q's respective potential sending time $t_{\text{send}M_p}$, the following relation holds:

$$StateAge(t_{\text{prune}}, Counter_{M_p} + 1) \leq StateAge(t_{\text{send}M_p}, Counter_{M_p} + 1) \leq age\frac{1+\rho}{1-\rho} +$$
$$\delta(1+\rho) + \pi(1+\rho) + \delta(1+\rho) + \pi(1+\rho) = age\frac{1+\rho}{1-\rho} + 2d(1+\rho) \quad .$$

□

Theorem 7 *Any message,* M_p*, sent by a correct node p will be assessed as timely by all correct receiving nodes,* q[6].

Proof: Let M_p be sent by a correct node p, and received by a correct node q at time $t_{arr.}$

- Timeliness Condition 1: $0 \leq Counter_{M_p} \leq n-1$ as implied by Lemma A.1 and by the fact that the CS cannot hold a negative number of signature entries.
- Timeliness Condition 2: Following Lemma A.1 a correct node cannot fire during the absolute refractory period. Restriction 5 therefore implies that a correct node cannot count less than $\tau(n+3)$ time units between its consecutive firings. A previous message from a correct node

[6] Theorem 7 might not be correct within a "decay" long time interval after initialization of the last node during the start up of the system. The reason is that until initialization of all the correct nodes they will not necessarily keep the same set of received messages and thus equal a state in which there is a violation of the assumption boundaries. Any message received after a "decay" long time interval after initialization of the last node has necessarily been received by all correct nodes and thus from this point on Theorem 7 holds.

will therefore be at least $\tau(n+2)$ time units old at any other correct receiving node before it will receive an additional message from that same node. Following the PRUNE procedure, the former message will therefore have decayed at all correct receiving nodes and therefore cannot be present in the message state at the arrival time of the subsequent message from the same sender.

- Timeliness Condition 3: This timeliness condition validates $Counter_{M_p}$. The validation criterion relies on the relation imposed at the sending node by the PRUNE procedure, between the $MessageAge$ of its accounted signature entries and its current Counter. The receiving node compares $Counter_{M_p}$ to the $MessageAge$ of its own signature entries, allowing for an additional network delay time and taking into consideration a possible clock skew (see Table **??**). Allowing for the clock skew and the network delay time of every signature entry accounted for in $Counter_{M_p}$, the largest message age can be formulated as:

$$[\ldots[[2d(1+\rho)]\frac{1+\rho}{1-\rho}+2d(1+\rho)]\frac{1+\rho}{1-\rho}+2d(1+\rho)]\cdots])]\frac{1+\rho}{1-\rho}+2d(1+\rho)],$$

which for a $Counter_{M_p}$ of k sums up to

$$2d(1+\rho)\pi(\frac{1+\rho}{1-\rho})^k+2d(1+\rho)\pi(\frac{1+\rho}{1-\rho})^{k-1}+\cdots+2d(1+\rho)\pi(\frac{1+\rho}{1-\rho})+2d(1+\rho)$$

$$=2d(1+\rho)\sum_{i=0}^{k}(\frac{1+\rho}{1-\rho})^i=2d(1+\rho)\frac{(\frac{1+\rho}{1-\rho})^{k+1}-1}{(\frac{1+\rho}{1-\rho})-1}\equiv\tau(k)$$

Corollary A.5 $\tau(k-1)\frac{1+\rho}{1-\rho}+2d(1+\rho)=\tau(k)$.

This gives way for the comparison criterion of Timeliness condition 3.

The proof is by induction on $Counter_{M_p}$, denoted by k. We refer to the first condition in Timeliness Condition 3 as a. and the second as b.

For $k=0$ condition a: The signature entry of p was inserted into the UCS of q at Timeliness Condition 2. The PRUNE Procedure ensures that this signature entry will not be in RUCS at the time of execution of Timeliness Condition 3, thus $|\text{message state}|\geq 1$. Hence, $Counter_{M_p}=0<1\leq|\text{message state}|$.

For $k=0$ condition b: q does not wait for any message to arrive and thus at q's potential sending time of M_q, $t_{\text{send}M_p}$, $State_{A}ge(t_{\text{send}M_p},1)$ will be at most π, the bound on the time between t_{arr} and $t_{\text{send}M_p}$. Hence, following Lemma A.4 at q's prune time t_{prune} and at $t_{\text{send}M_p}$:

$$StateAge(t_{\text{prune}},1)\leq StateAge(t_{\text{send}M_p},1)\leq\pi<2d(1+\rho)=\tau(0).$$

For $k=1$ condition a: q will receive and process the signature entry accounted for in M_p within $d(1+\rho)$ time units of $t_{\text{send}M_p}$ at p. Lemma A.2 ensures that this signature entry will not be in RUCS. Thus, $Counter_{M_p}=1<2\leq|\text{message state}|$ from some time t' in the interval $[t_{arr},t_{arr}+d(1+\rho)]$ and at least until the end of the interval, which ensures that q will perform the test successfully.

For $k=1$ condition b: Following the PRUNE procedure at p, the oldest signature entry accounted for in $Counter_{M_p}$ was at most π time units old at p (on p's clock) at time $t_{\text{send}M_p}$. Following Lemma A.4 at q's prune time t_{prune}, and potential sending time, $t_{\text{send}M_p}$:

$$StateAge(t_{\text{prune}},2)\leq StateAge(t_{\text{send}M_p},2)\leq\pi\frac{1+\rho}{1-\rho}+2d(1+\rho)<2d(1+\rho)\frac{1+\rho}{1-\rho}+2d(1+\rho)=\tau(1).$$

For $k = 2$ condition a: Node q must receive and process the two signature entries accounted for in M_p within $d(1+\rho)$ time units of t_{send} at p. Lemma A.2 ensures that these signature entries will not be in RUCS and thus $Counter_{M_p} = 2 < 3 \leq |\text{message state}|$ from some time t' in the interval $[t_{arr}, t_{arr} + d(1+\rho)]$ and at least until the end of the interval.

For $k = 2$ condition b: Following the PRUNE procedure, the oldest signature entry accounted for in $Counter_{M_p}$ was at most $\tau(1)$ time units at p at time t_{prune}, the time it executed the PRUNE procedure. Following Lemma A.4 at q's prune time t_{prune}, and potential sending time, $t_{\text{send}M_p}$:

$$StateAge(t_{\text{prune}}, 3) \leq StateAge(t_{\text{send}M_p}, 3) \leq \tau(1)\frac{1+\rho}{1-\rho} + 2d(1+\rho) = \tau(2).$$

For $2 < k < n$ condition a: q must receive and process the k signature entries accounted for in M_p within $d(1+\rho)$ time units of $t_{\text{send}M_p}$ at p. Following Lemma A.2 these k signatures are present in q's message state. Hence, $Counter_{M_p} = k < k+1 \leq |\text{message state}|$ from some time t' in the interval $[t_{arr}, t_{arr} + d(1+\rho)]$ and at least until the end of the interval.

For $2 < k < n$ condition b: Let us consider the case where $Counter_{M_p}$ accounts for a signature entry from a faulty node such that q still holds a previous signature entry from this faulty node in its message state. Furthermore, let M_p arrive before the new message from the faulty node arrives at q. If this scenario causes a violation of Timeliness Condition 3, q will wait up to $d(1+\rho)$ time units. The faulty node's new message will arrive within this time window and cause the previous signature entry to be deleted.

Following the PRUNE procedure at p, the oldest of the k signature entries accounted for in $Counter_{M_p}$ was at most of age $\tau(k-1)$ at p at time t_{prune}, the time it executed the PRUNE procedure. Following Lemma A.4 at q's prune time t_{prune}, and the potential sending time, $t_{\text{send}M_p}$:

$$StateAge(t_{\text{prune}}, k+1) \leq StateAge(t_{\text{send}M_p}, k+1) \leq \tau(k-1)\frac{1+\rho}{1-\rho} + 2d(1+\rho) = \tau(k).$$

We therefore proved that Timeliness Condition 3 holds for any $0 \leq k < n$.

The message M_p will therefore be assessed as timely by all correct receiving nodes. □

Lemma A.6 *Following the arrival and assessment of a timely message M_p at node q, the subsequent execution of the* MAKE-ACCOUNTABLE *procedure yields $Counter_q > Counter_{M_p}$.*

Proof: We first show that at time t, the time of execution of the MAKE-ACCOUNTABLE procedure, $\max(1, Counter_{M_p} - Counter_q + 1) \leq |\text{UCS}|$, ensuring the existence of a sufficient number of signature entries in UCS to be moved to CS.

M_p is assessed as timely at q, therefore, by Timeliness Condition 3 and Lemma A.2, at time t,

$$\begin{aligned}
Counter_{M_p} &< |\text{message state}| = |\text{CS}| + |\text{UCS}| = Counter_q + |\text{UCS}| = \\
&\quad Counter_{M_p} - \max(1, Counter_{M_p} - Counter_q + 1) + 1 + |\text{UCS}| \\
&\Rightarrow 0 < -\max(1, Counter_{M_p} - Counter_q + 1) + 1 + |\text{UCS}| \\
&\Rightarrow \max(1, Counter_{M_p} - Counter_q + 1) - 1 < |\text{UCS}| \\
&\Rightarrow \max(1, Counter_{M_p} - Counter_q + 1) \leq |\text{UCS}|.
\end{aligned}$$

There are two possibilities at the instant **prior** to the execution of the MAKE-ACCOUNTABLE procedure. The procedure requests exclusive write access to CS, therefore $Counter_q = |\text{CS}|$ at this instant:

1. $Counter_{M_p} \leq Counterq$, then $\max(1, Counter_{M_p} - Counter_q + 1) = 1$, meaning |CS| will increase by 1.

2. $Counter_{M_p} > Counter_q$, then |CS| will increase to $Counter_q + \max(1, Counter_{M_p} - Counter_q + 1) = Counter_q + Counter_{M_p} - Counter_q + 1 = Counter_{M_p} + 1$.

In either case, immediately subsequent to the execution of the procedure we get: $|\text{CS}| > Counter_{M_p}$ and therefore the updated $Counter_q > Counter_{M_p}$. $\square$

Lemma A.7 *Following the arrival of a timely message M_p, $Counter_q > Counter_{M_p}$ at time t_{sendM_p}.*

Proof: Let t_{arr} denote the arrival time of M_p at q. In the interval $[t_{arr}, t_{\text{send}M_p}]$ at least one PRUNE procedure is executed, the one which is triggered by the arrival of M_p. Following Lemma A.6, $Counterq > Counter_{M_p}$ subsequent to the execution of MAKE-ACCOUNTABLE procedure. By Lemma A.2 all signature entries accounted for in $Counter_{M_p}$ will not be moved out of the message state by any PRUNE procedure executed up to time $t_{\text{send}M_p}$, thus, *Counter* must stay with a value greater than $Counter_{M_p}$ up to time $t_{\text{send}M_p}$. $\square$

Theorem 8 *The* SUMMATION *algorithm satisfies Restriction 1.*

Proof: Let p denote a correct node that sends M_p. Theorem 7 ensures that M_p is assessed as timely at every correct receiving node. Lemma A.7 ensures that the value of *Counter* will not decrease below $Counter_{M_p} + 1$ until time $t_{\text{send}M_p}$, thereby satisfying Restriction 1. $\square$

A.4 Proof of Lemma 4.5

Recall that the statement of Lemma 4.5 is:
Let $p, q \in C_1$ and $r \in C_2$, denote three correct nodes belonging to two different synchronized clusters. Following the arrival and assessment of p's and q's fires, both will be accounted for in the *Counter* of r.
Proof: Let p fire before node q. Following Lemma 4.2 node q will fire within σ of p ($d(1+\rho)$ on r's clock). Node r will receive and assess q's fire at a time t_q at most $d(1+\rho)+d(1+\rho) = 2d(1+\rho)$ after p fired. Restriction 1 ensures that r will account for each one after their arrival and assessments. $MessageAge(t_q, p) \leq 2d.(1+\rho) = \tau(0)$ and therefore node r did not decay or move M_p to RUCS by time t_q. Therefore, M_p is still accounted for by node r at time t_q, and therefore $Counter_r \geq 2$ at time t_q. $\square$