OCD: Obsessive Consensus Disorder (or Repetitive Consensus)

Danny Dolev*
Hebrew University
dolev@cs.huji.ac.il

Ezra N. Hoch
Hebrew University
ezraho@cs.huji.ac.il

ABSTRACT

Consider a distributed system S of sensors, where the goal is to continuously output an agreed reading. The input readings of non-faulty sensors may change over time; and some of the sensors may be faulty (*Byzantine*). Thus, the system is required to repeatedly perform consensus on the input values.

This paper investigates the following question: assuming the input values of all the non-faulty sensors remain unchanged for a long period of time, what can be said about the agreed-upon output reading of the entire system? We prove that no system's output is stable, i.e. the faulty sensors can force a change of the output value at least once.

We show that any system with binary input values can avoid changing its output more than once, thus matching the lower bound. For systems with multi-value inputs, we show that the output may change at most twice; when $n = 3f + 1$ this solution is shown to be tight. Moreover, the solutions we present are self-stabilizing.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Reliability, Theory

Keywords

Distributed computing, fault tolerance, self-stabilization, Byzantine failures, repetitive consensus, long-lived consensus.

*This work was funded in part by ISF.

1. INTRODUCTION

Consensus is one of the fundamental problems in distributed algorithms, and a vast body of literature exists on the subject (see [8], [10] and [1]). The "common" consensus problem consists of nodes with static input values that are required to agree on some output value within a finite time. Different extensions of the consensus problem study different directions; in this paper we consider the case where a sequence of multiple consensuses takes place.

Executing multiple consensus comes in different flavors; *Continuous Consensus* aims at continuously agreeing on the history of inputs of all nodes in the system (see [12]). *Multi-consensus* efficiently executes multiple consensuses sequentially (see [2]), *Committee Decision* executes multiple consensuses concurrently (see [9]); and others, such as [13]. We are interested in a different aspect of executing multiple consensuses.

When executing m consensuses, there are m agreed-upon output values. This raises the following question: What is the connection between the different outputs of the different consensuses? Is there any way to guarantee that consecutive consensuses output the same value - i.e. that the output of the different consensuses is stable? None of the above papers has investigated the stability of the output of the different consensuses in the sequence. The first paper to discuss the stability of "long-lived" consensus is [7] (from which the term "long-lived" is taken). [7] (and later, [4]) also defines measurements to estimate solutions of the "long-lived" consensus problem; in the current work, we consider similar measurements (see Section 2).

Consider a system consisting of n nodes, where each node receives an input value from the set V, $V = \{0, \ldots, v-1\}$; these input values can change over time, and hence the nodes need to repeatedly agree on their collective output value. For example, consider a network of sensors that needs to agree on a single output: assume some of the nodes are malfunctioning (*Byzantine*), and consider the following question: if the input of the nodes changes overtime and eventually stabilizes, is it possible to ensure that the output will also stabilize eventually? As will be shown below, the answer is "No". However: if the input does not change, then (under certain restrictions) the output can change at most once. For example, assume a system with $V = \{0, 1\}$ in which at some time the input values of all non-faulty nodes have stopped changing. If no faulty nodes exist, the output will eventually stop changing. However, we show that even a single *Byzantine* node can delay this transition indefinitely.

The above impossibility result is true for any system with

Byzantine presence, regardless of the communication model (synchronous, asynchronous, etc). However, our solution to reducing output changes operates in a highly-fault tolerant manner: it is self stabilizing[1] and *Byzantine* tolerant. That is, starting from any initial state, in the presence of permanent f *Byzantine* nodes, the number of output changes is bounded.

Related work: Overcoming faults in sensors by averaging the input values has been done in [11]; however the averaging is done by one node connected to all sensors, and has low fault resiliency.

Quantifying faulty nodes' effect on the system's output was investigated in [7] and [4]. These papers consider a *geodesic path*, which is a list of input vectors (representing the inputs to each node) in which each node changes its input value at most once. The two papers concentrate on the stability assurances that can be guaranteed for geodesic paths. However, they do not consider the *Byzantine* case, in which a small set of faulty nodes repeatedly face changes in their input values.

Recently, [15] has extended [7]'s worst-case scenario results to the average case, as well as showing the importance of memory for the output's stability. [15] considers paths that are randomly selected, and thus examines the average case scenario. However, [15] does not incorporate *Byzantine* behavior, which coordinates its failures in a malicious (and non-random) fashion.

[4] and [7] show that for memory-less consensus functions, even with a single faulty node, the output may change infinitely many times. However, the behavior of *Byzantine*-tolerant consensus functions with memory has not been investigated yet. In addition, the results of [7], [4] and [15] have not been shown to be self-stabilizing.

Contribution: We define an *oblivious path*, which is a list of input vectors in which at most f nodes change their input values (arbitrarily many times). We then quantify a system output's stability for such paths, giving both lower and upper bounds. The solution is shown to be self-stabilizing.

2. COMPUTATIONAL MODEL

Following [4] and [7], we begin this paper with an analysis that ignores the communication and timing model of the system. Let $V, V = \{0, \ldots, v-1\}$ be the set of allowed input values, and let $\mathcal{F} : V^n \to V \cup \{\perp\}$ be a function on an input vector $\vec{x} \in V^n$. Given a list of input vectors $\vec{x_0}, \vec{x_1}, \ldots, \vec{x_l}$, we are interested in the behavior of $\mathcal{F}(\vec{x_0}), \mathcal{F}(\vec{x_1}), \ldots, \mathcal{F}(\vec{x_l})$. To allow $\mathcal{F}$ to have memory (adding states to the system) we consider $\mathcal{F} : V^n \times M \to V \cup \{\perp\}$, and a state transition function $\tau : V^n \times M \to M$.

Intuitively, an input vector $\vec{x} \in V^n$ represents the inputs of each node in the distributed system, while the state transition function τ represents the change in the state of the distributed system (sending/reciving messages and their effects on each node's internal memory). Lastly, $\mathcal{F}$ represents the output of all non-faulty nodes in the system. The formal definitions are as follows (based on those of [4]):

DEFINITION 2.1. *A* **system** *is defined by a 6-tuple* $S = \langle n, v, f, M, \tau, \mathcal{F} \rangle$ *, where* n *is the number of nodes,* v *is the size of the input space* $V = \{0, \ldots, v-1\}$*,* f *is a bound on the number of faulty nodes,* M *is a set of memory states,* τ *is a state transition function* $\tau : V^n \times M \to M$ *and* $\mathcal{F}$ *is the output function* $\mathcal{F} : V^n \times M \to V \cup \{\perp\}$*.*

For the above definition to be "interesting" a "*validity*" requirement on $\mathcal{F}$ is added: the output of $\mathcal{F}$ must be the input of some correct node. Since $\mathcal{F}$ does not "know" which nodes are faulty and which are not, it can output a value ν only if it has seen at least $f+1$ copies of ν in the input vector; otherwise it must output $\perp$.

We also require that if some value ν' appears $n-f$ (or more) times in the input vector, then the output of $\mathcal{F}$ is ν'. More formally, let $\#\nu(\vec{x})$ be the number of occurrences of ν in $\vec{x}$: if $\mathcal{F}(\vec{x}) = \nu \in V$ then $\#\nu(\vec{x}) \geq f+1$; if $\mathcal{F}(\vec{x}) = \perp$ then $\forall_{\nu \in V} \#\nu(\vec{x}) < n-f$.

The above *validity* is considered throughout the paper. In Section 6 a different *validity* measure is presented.

DEFINITION 2.2. *An* **input path** *(path for short)* P *(of some system* S*) is a list of input vectors* $\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{l-1} \in V^n$*. Component* k *is said to* **change** *on path* P *if there is some input vector* $\vec{x}_i$ *such that* $\vec{x}_i[k] \neq \vec{x}_{i+1}[k]$*, where* $\vec{x}[k]$ *is the* k*th entry in the vector* $\vec{x}$*.*

DEFINITION 2.3. *A* **run** *of a system* S *from memory state* m_0 *on some path* P *is the sequence* $m_1 = \tau(\vec{x}_0, m_0), m_2 = \tau(\vec{x}_1, m_1), \ldots, m_l = \tau(\vec{x}_{l-1}, m_{l-1})$*;*

The **decision output** *of the above run is the sequence* $\mathcal{F}(\vec{x}_0, m_0), \mathcal{F}(\vec{x}_1, m_1), \ldots, \mathcal{F}(\vec{x}_{l-1}, m_{l-1})$*. Denote by* $O_i := \mathcal{F}(\vec{x}_i, m_i)$*. The decision output changes at index* $i > 0$ *if* $O_{i-1} \neq O_i$*.*

[4] and [7] define a **geodesic input path** as an input path in which each component changes at most once. In addition, they define the **instability** of a system S as the maximal number of decision output changes for any geodesic input path, starting from S's initial state. In the current work we are interested in a somewhat different stability measurement, so we henceforth refer to the instability of [4] and [7] as **geodesic-instability**, in order to differentiate it from our usage of the term "instability".

Note that *Byzantine* nodes may "pretend" that their inputs are consistently changing. It is therefore important to design systems that are robust in the face of such behavior, *i.e.* , that do not change their output if the correct nodes' inputs do not change. In other words, the target should be a system that is oblivious to changes in *Byzantine* nodes' input values (as long as the other nodes have stable input values). Hence, we define the following:

DEFINITION 2.4. *An* **f-oblivious input path** *(oblivious path, for short) of a system* S *is an input path* P *in which at most* f *components change.*

Notice that a geodesic path's length is bounded by n, since each of its components can change at most once. However, an oblivious path can be arbitrarily long, since f components can repeatedly change. Faulty nodes can behave arbitrarily at any point during the execution of S. Moreover, the input values of non-faulty nodes may change due to external readings. Thus, we aim at expressing the robustness with respect to all memory states and to all oblivious paths.

DEFINITION 2.5. *Let* $CT(S, m, P)$ *be the (possibly infinite) number of times the decision output of* S *changes when*

[1]For more on self-stabilizing see [14].

running on oblivious path P from memory state m. Let $MaxCT(S,m) := max_P\{CT(S,m,P)\}$.

Similarly, $CL(S,m,P)$ is the (possibly infinite) largest index for which the decision output of S changes when running on oblivious path P from memory state m.
Let $MaxCL(S,m) := max_P\{CL(S,m,P)\}$.

DEFINITION 2.6. *The* **count-instability** *of a system S is the maximal number of decision output changes during a run from any memory state m and for any oblivious input path P. Formally, we can define count-instability as: $max_m\{MaxCT(S,m)\}$.*

The **length-instability** *of a system S is the smallest index after which no change occurs to the decision output during a run from any memory state m and for any oblivious input path P. Formally, length-instability can be expressed as: $max_m\{MaxCL(S,m)\}$.*

It has been shown (in [4], [7]) that in memory-less systems S (systems in which $M = \{m_0\}$) in which some component may change its input infinitely many times, a path can be constructed to cause any number of decision output changes. In other words, for memory-less S with $f \geq 1$, the count-instability of S is ∞. This is the reason that [4] and [7] discuss only geodesic-instability, but do not discuss count-instability or length-instability at all. An interesting extension to the above oblivious-path definition (and therefore to the count/length instability definitions) that is not discussed in the current paper is one which considers changes to input values of non-faulty nodes. Notice that the lower bounds of Section 3 hold for this extension as well.

Table 1 presents the lower and upper bounds appearing in the following sections.

Table 1: Summary of results

parameters	count-instability	theorem#
$f > 0$	≥ 1	Theorem 1
$n \geq 3f+1$	≤ 2	Theorem 4
$n = 3f+1, \lvert V\rvert > 2$	≥ 2	Theorem 3
$n \geq 3f+1, \lvert V\rvert = 2$	1	Theorem 5

2.1 A General System Model

The computational model presented above captures common requirements of any distributed system that tries to repeatedly reach consensus on changing input values. Therefore, any lower bound presented in this computational model holds for any distributed system.

A distributed system T may have richer semantics than the proposed computational model. However, the properties of the computational model encompass the very basic behavior of repeatedly reaching consensus; therefore, it should be possible to "embed" the computational model in T's semantics. That is, in certain scenarios (such as when restricting the behavior of T) there is a reduction from T to the computational model.

For example, suppose that T is a sparsely connected network of nodes that gathers inputs from different nodes, waits until they do not change (to avoid fluctuations in the output), propagates them throughout the system, and only then decides on an output. In any system, on a run of T, at some time t_{-1}, the input will change to $\vec{x}_{-1}$ and stay there. Eventually, T outputs some value O_{-1} at time t_0. Consider m_0 to be the state of the system at t_0. At some time $t'_0 \geq t_0$ the input changes to $\vec{x}_0$ and eventually - at time t_1 - the system outputs O_0. Mark by m_1 the state of the system at t_1. In general, at time t_i the system is at some state m_i, at time $t'_i \geq t_i$ the input changes to $\vec{x}_i$ and at time $t_{i+1} \geq t'_i$ the system outputs a value O_i; the run continues in this manner.

Consider the scenario where $t'_i = t_i$ for all i; that is, the input changes immediately after the output has been determined. T must also operate correctly in the described scenario. However, in such a setting the following reduction can be used: $m_0, m_1, \ldots$ will be the states of the system; $\vec{x}_0, \vec{x}_1, \ldots$ will be the input vectors; τ will denote the system's state change between 2 consecutive outputs; and $\mathcal{F}$ will denote the system's output, given the previous system's state and the new input. Notice that T might change its state more than once between t_i and t_{i+1}; however, we are only interested in the change of state from the state at t_i (m_i) to the state at t_{i+1} (m_{i+1}).

The above example demonstrates how a system that produces outputs every so often, in accordance to inputs that change over time, has scenarios in which it can be abstracted using the computational model. This observation is the motivation behind using the computational model and the lower bounds proven in this paper.

3. IMPOSSIBILITY RESULTS

THEOREM 1. *For any system S with $f \geq 1$ the count-instability of S is at least 1.*

PROOF. Consider the path P : $\vec{x}_0 := 0^{n-f}1^f, \vec{x}_1 := 0^{n-f-1}1^{f+1}, \ldots, \vec{x}_i := 0^{n-f-i}1^{f+i}, \ldots, \vec{x}_{n-2f} := 0^f1^{n-f}$. Let m_0 be some state of S and consider S's run on P from m_0. Denote the decision output of S's run from m_0 on P as $O_0 := \mathcal{F}(\vec{x}_0, m_0), O_1 := \mathcal{F}(\vec{x}_1, \tau(m_0, \vec{x}_0))$, etc. Denote by m_i the ith memory state of the above run; that is $m_{i+1} = \tau(\vec{x}_i, m_i)$.

By *validity*, since $\#0(\vec{x}_0) \geq n - f$ it holds that $O_0 = 0$, and for similar reasons $O_{n-2f} = 1$. Let j be the first O_j that is 1; more formally let j be $j = min\{i|\mathcal{F}(\vec{x}_i, m_i) = 1\}$ (note that $j \geq 1$). Consider the memory state m_{j-1}: when S is at state m_{j-1}, given the input vector $\vec{x}_{j-1}$, S's output is 0 and it moves to state m_j. Then, for $\vec{x}_j$ as input, S's output is 1. Thus, the path $P := \vec{x}_{j-1}, \vec{x}_j$ is a path that causes S's run starting from m_{j-1} to change once. Notice that P is an oblivious path (for $f > 0$), since $\vec{x}_{j-1}, \vec{x}_j$ differ by exactly one component.

This implies that for any system S (with $f > 0$) there exists an oblivious path P and a state m such that S's run starting from m on P changes its decision output at least once. And so,S's count-instability is ≥ 1. □

Theorem 1 states that any system that is designed to be tolerant of even a single *Byzantine* node, must have states from which the *Byzantine* nodes can change the decision output value. The following theorem shows that the *Byzantine* node(s) can delay this output change indefinitely.

THEOREM 2. *For any system S with $f \geq 1$ the length-instability of S is ∞.*

PROOF. Let $\vec{x}_i := 0^{n-f-i}1^{f+i}$ and consider the path

$$P^k := \underbrace{\vec{x}_0, \ldots, \vec{x}_0}_{k}, \ldots, \underbrace{\vec{x}_i, \ldots, \vec{x}_i}_{k}, \ldots, \underbrace{\vec{x}_{n-2f}, \ldots, \vec{x}_{n-2f}}_{k} .$$

P^k is a path consisting of $n-2f+1$ "sections", each of length k. Each such section consists of a single input vector, where the ith section consists of k repetitions of $\vec{x}_i$. P^k is $k\cdot(n-2f+1)$ vectors long. Denote by $\vec{x}^j$ the jth vector in P^k ($j \in [0, k\cdot(n-2f+1)]$); note that $\vec{x}^j = \vec{x}_{\lfloor \frac{j}{k} \rfloor}$. For a given j, let m_j be the memory state before the input vector $\vec{x}^j$ is processed, and let O_j be the decision output of $\mathcal{F}(m_j, \vec{x}^j)$.

Due to *validity*, since $\#0(\vec{x}_0) \geq n-f$ it holds that $O_0 = 0$ and $O_j = 0$ for $j < k$. For similar reasons $O_{k\cdot(n-2f)} = 1$. Therefore, $O_j \neq O_{j-1}$ for some $j \geq k$. Consider the first such j. Denote the path $P := \vec{x}^{j-k}, \vec{x}^{j-k+1}, \ldots, \vec{x}^j$; P is an oblivious path, since at most one component changes in P. When running S from memory state m_{j-k} on the path P the output starts as "0" and changes to "1" after k input vectors; that is, $CL(S, m_{j-k}, P) \geq k$.

Notice that the above proof is independent of k's value. Thus for any k, there exists a memory state m and an oblivious path P such that $CL(S, m, P) \geq k$. That is, the length-instability of S is ∞. □

Theorem 2 implies that even a single *Byzantine* node can cause a decision output change in a system "whenever it wants". More specifically, there is a state of the system such that even if the inputs of all non-*Byzantine* nodes do not change, there is no bound on when a *Byzantine* node can cause the decision output of the system to change.

THEOREM 3. *For any system S with $n = 3f+1$ and $|V| \geq 3$ the count-instability of S is at least 2.*

PROOF. Let m_0 be some initial state and let $\vec{x} := 0^{2f+1}1^f$. Due to *validity*, $\mathcal{F}(\vec{x}, m_0) = 0$. Now, consider the vector $\vec{y} := 0^{f+1}1^{f+1}2^{f-1}$ and $m_1 := \tau(\vec{x}, m_0)$. $\mathcal{F}(\vec{y}, m_1)$ can be "0", "1" or $\perp$.

If it is "1" or $\perp$, then the path $P := \vec{x}, \vec{y}, \vec{x}$ causes S to change its output twice.

Similarly, assume that $\mathcal{F}(\vec{y}, m_1) = 0$ and let $m_2 := \tau(\vec{y}, m_1)$. Let $\vec{z} := 0^f 1^{f+1} 2^f$. Now, $\mathcal{F}(\vec{z}, m_2)$ can be either "1" or $\perp$. If it is $\perp$ then the path $P := \vec{y}, \vec{z}, 0^f 1^{2f+1}$ causes the system S to change its output twice. On the other hand, if $\mathcal{F}(\vec{z}, m_2) = 1$ then the path $P := \vec{y}, \vec{z}, 0^f 1^f 2^{f+1}$ changes the output twice. Thus, there exists an f-oblivious path that causes S to change its output twice. □

REMARK 3.1. *The $|V| \geq 3$ limit (in Theorem 3) is required. Moreover, if $|V| = 2$ then it is possible to construct a system S that has count-instability of 1. In Theorem 5 we show how to achieve exactly that.*

4. REPETITIVE CONSENSUS

In the above sections a formal analysis of the computational model was given, together with lower bounds regarding the *count-instability* of any distributed system. In the rest of the paper we concentrate on a more "practical" approach; that is, the repetitive consensus problem is defined and solved in a "synchronous" network in a self-stabilizing and *Byzantine* tolerant manner.

We start by defining the Global-Beat-System (a self-stabilizing equivalent of the classical synchronous network); then we present and solve the repetitive consensus problem as applied to this model.

4.1 Model and Definitions

The "Global Beat System" model (GBS for short) consists of n nodes that can communicate via message passing, where the sender of each message can be identified. Nodes have access to a common "global beat system", which produces signals/beats at regular intervals, such that a message sent by any node p to any node q upon receiving some beat, reaches q before the following beat. All nodes receive a signal/beat at the same time, and can perform computations and/or send messages upon its receipt. The beats are spaced in such a way as to allow a correct node to send one message to each correct node (and process such messages) in the time span between two consecutive beats.

In non-self-stabilizing synchronous systems, usually there is a common counter that all correct nodes are aware of (the current round number). This is not the case in the GBS model; however, for the sake of clarity we will refer to an "external" beat/round number r, that the nodes are not aware of. In the rest of this paper, "rounds" and "beats" will be used interchangeably. These terms are not to be confused with "pulses", which refer to the output of pulsing algorithms.

While the system is "unstable" (due to transient faults), any number of nodes may behave in a *Byzantine* manner and the communication network may behave arbitrarily. However, once the system "stabilizes", there will be at most f *Byzantine* nodes, the global beat system will produce beats regularly and the communication network will deliver messages on time (before the following beat).

DEFINITION 4.1. *A node is* **non-faulty** *when it follows the given protocol*[2]*, and* **faulty** *otherwise.*

REMARK 4.1. *In the current work processing time of incoming messages is ignored. This assumption does not weaken the result, it only simplifies the exposition.*

DEFINITION 4.2. *The communication network is* **non-faulty** *when the following conditions hold:*

1. *A message by a correct node p sent upon a receipt of a beat from the global beat system, arrives (and is processed) at its destination before the following beat is issued by the global beat system;*
2. *The sender's identity and the message context of any message received are not tampered with.*
3. *A message received by p was sent by some node no more than one beat ago. That is, "phantom" messages are not delivered.*

In real-world networks, it may take some time for the communication network to overcome transient faults. Specifically, the communication networks' buffers may contain messages that were not sent by any node, and the network may eventually deliver them. We consider the communication network to be *non-faulty* only after all of these "phantom" messages have been delivered or cleared away.

According to the above definition, once the network is non-faulty, it adheres to the GBS model. Which means that

[2]Notice that a non-faulty node p may exhibit unwanted behavior, due to its arbitrary state. However, given p's state, its behavior is determined by the protocol.

messages cannot be lost and old messages cannot be stored for an arbitrarily long time.

The transition from being faulty to becoming a "valid" participant of the protocol can't be instantaneous. Therefore, a continuous period of non-faulty behavior is required before the system or a node can be considered correct.

DEFINITION 4.3. *A node is* **correct** *following Δ_{node} beats of continuous non-faulty behavior in which the communication network is non-faulty.*

DEFINITION 4.4. *The system is* **coherent** *when the communication network is non-faulty and there are $n-f$ correct nodes.*

The values of n and f are fixed constants and are considered part of the "code" of the protocols and thus non-faulty nodes cannot initialize with arbitrary values for these constants.

4.2 The Repetitive Consensus Problem

Let I_p^r and O_p^r be the input and output value of p at beat r. Input values are from some finite set V, and output values are from $V \cup \{\perp\}$. Let $\mathcal{G}^r$ be the set of non-faulty nodes at beat r; (we will use $\mathcal{G}$ when r is clear from the context.)

DEFINITION 4.5. *The inputs (of non-faulty nodes) are* **stable** *during $[r_1, r_2]$ if for every node $p \in \mathcal{G}$ the value of I_p^r does not change, for all $r \in [r_1, r_2]$. This condition can be expressed formally as: $\forall_{p\in\mathcal{G}} \forall_{r_1 \le r \le r_2} [I_p^r = I_p^{r_1}]$.*

Similarly, we say that the outputs are stable during $[r_1, r_2]$ if $\forall_{p\in\mathcal{G}} \forall_{r_1 \le r \le r_2} [O_p^r = O_p^{r_1}]$.

Let I^r denote the vector of inputs of all nodes $I^r := (I_{p_1}^r, \ldots, I_{p_n}^r)$ and let O^r denote the vector of outputs at beat r. When talking about outputs we consider only outputs of non-faulty nodes, since no requirements can be given on outputs of *Byzantine* nodes. Likewise, only non-faulty nodes' inputs are considered, as a *Byzantine* node can have any input it wishes.

DEFINITION 4.6. *The outputs are* **in agreement** *at some beat r if $\forall_{p,p'\in\mathcal{G}} [O_p^r = O_{p'}^r]$; denote by $\mathcal{V}^r$ the agreement value and let $\mathcal{V}^r := O_p^r$ for some $p \in \mathcal{G}$.*

The outputs are **decisive** *during beat interval $R = [r_1, r_2]$ if the outputs are stable during R and the outputs are in agreement during beat r_1. Denote by $\mathcal{V}^{[r_1,r_2]}$ the agreement value and let $\mathcal{V}^{[r_1,r_2]} := \mathcal{V}^{r_1}$.*

For the outputs to become stable, the inputs must not change for a "long-enough" period of time, leading to the following definition:

DEFINITION 4.7. *A beat interval $R = [r_1, r_2]$ is* **Δ-applicable** *if $r_2 - r_1 \ge \Delta$ and the inputs are stable during R. We use the notation $R_{|\Delta} := [r_1 + \Delta, r_2]$.*

Hopefully, there exists some Δ, s.t. for any Δ-applicable interval R, the outputs are decisive in $R_{|\Delta}$. That is, if the inputs do not change for long enough, then as long as they continue not to change, the outputs are in agreement and do not change. However, due to Theorem 1 and Theorem 2 this is impossible; we therefore add the following definitions:

DEFINITION 4.8. *The outputs are* **k-decisive** *during beat interval $R = [r_1, r_2]$ if the outputs are in agreement during R; and R consists of k disjoint intervals $R_1, \ldots, R_k$, such that: a. $\bigcup R_i = [r_1, r_2]$; b. for each interval R_i the outputs are stable.*

DEFINITION 4.9. *A system* **k-changes its mind** *in the interval $[r_1, r_2]$ if k is the minimal value such that the outputs are $(k{+}1)$-decisive. Alternatively, we say that the system changes its mind k times.*

Notice that "decisive" as defined in Definition 4.6 is the same as "0-changes its mind" as defined in Definition 4.9.

The system is considered to uphold (k, Δ)-repetitive consensus behavior, if (for Δ-applicable intervals) the non-faulty nodes agree on their output, "*validity*" holds, and the output does not change more than k times. The following is a formal definition:

DEFINITION 4.10. *A system upholds a* **(k, Δ)-repetitive consensus behavior** *during interval R, if for any Δ-applicable interval $R' \subset R$:*

1. **Agreement:** *The outputs are in agreement in $R'_{|\Delta}$;*
2. **Validity:** *For any beat $r \in R'_{|\Delta}$, if $\mathcal{V}^r \neq \perp$ then some non-faulty node has $\mathcal{V}^r$ as its input value (during R'); if all non-faulty nodes have the same input value ν (during R'), then $\mathcal{V}^r = \nu$;*
3. **Bounded changes:** *The system changes its mind during $R'_{|\Delta}$ at most k times.*

Note that the above definition is applicable only for R's larger than Δ. Definition 4.10 defines "desired behavior", in the context of the repetitive consensus problem. It refers to any subinterval $R' = [r_1, r_2]$ of length at least Δ, saying that if the inputs are stable throughout R' then the following holds: "*agreement*" states that all outputs in the interval $[r_1 + \Delta, r_2]$ are the same at all non-faulty nodes. "*validity*" extends the definition from Section 2 to the distributed synchronous model at hand. Lastly, "*bounded changes*" states that the output of the system (in the interval $[r_1 + \Delta, r_2]$) may change at most k times. The motivation behind such a definition is straightforward: if the inputs are stable for long enough, then the system agrees on its output, the output is related (in a reasonable way) to the input, and the output does not change more than k times.

DEFINITION 4.11. *The* **k-repetitive consensus problem** *(k-RC for short) is solved by an algorithm $\mathcal{A}$ if there exists a constant Δ such that for any interval R the system upholds a (k, Δ)-repetitive consensus behavior.*

DEFINITION 4.12. *The* **self-stabilizing k-repetitive consensus problem** *(k-SSRC for short) states that: there exist constants $\Delta, \Delta_{stabilize}$, s.t. for any interval $[r_1, r_2]$ with no transient faults, where $r_2 - r_1 \ge \Delta_{stabilize}$, the system upholds the (k, Δ)-repetitive consensus behavior in the interval $[r_1 + \Delta_{stabilize}, r_2]$. $\Delta_{stabilize}$ is called the* **convergence time**.

The distributed model presented above is a weaker model (has less assumptions) than the computational model of

Algorithm SS-REP-CONS /* executed at node p */
/* $\mathcal{P}$ is a self-stabilizing pulse with $Cycle > \Delta_{agree}$[a] */

On beat: (signal from global beat system)

1. **Execute** a single beat of $\mathcal{P}$;
2. **If** $\mathcal{P}$ invoked a PULSE at the current beat:
   /* "vals" represents the agreed input vector */
   (a) **If** *vals* contains (at least) $n - f$ same value ν then set $Output_p := \nu$;
   Else if $Output_p$ does not appear $f + 1$ times in *vals*, set $Output_p := \perp$;
   (b) **Start** executing *Byzantine* agreement on I_p; /* I_p is p's current input */
   (denoted by BYZ-AGREE$_p$)
   (c) **Start** executing *Byzantine* consensus on $Output_p$;
   (denoted by BYZ-CONS)
   /* vals[q] holds q's input value */
3. **Once** a q's BYZ-AGREE$_q$ agreement terminates with v, update $vals[q] := v$;
4. **Once** the BYZ-CONS consensus terminates with v', update $Output_p := v'$;
5. **Set** $O_p := Output_p$; /* O_p is p's current output */

[a] Δ_{agree} is the number of rounds to execute *Byzantine* agreement.

Figure 1: An algorithm that solves the 2-SSRC problem in the GBS model.

Section 2. Therefore, the lower bounds of Section 3 hold for the distributed model as well. [3]

COROLLARY 1. *The 0-RC problem cannot be solved; The 0-SSRC problem cannot be solved, for any value of* $\Delta_{stabilize}$.

PROOF. Immediate from Theorem 1, Theorem 2 and the discussion in Section 2.1. □

REMARK 4.2. *Notice that the first part of the above corollary holds for both self-stabilizing and non-self-stabilizing models. That is, the 0-RC problem cannot be solved even in a non-self-stabilizing synchronous model in the presence of even a single* Byzantine *fault.*

5. SOLVING SS-REPETITIVE CONSENSUS

It is impossible to solve the 0-SSRC problem (or even the 0-RC problem) for any value of $f > 0$; in addition, for $n = 3f+1, |V| \geq 3$ it is impossible to solve the 1-SSRC (and the 1-RC) problem. In the following section we present a solution for the 2-SSRC problem for any value of $n \geq 3f+1$ (see Figure 1).

The SS-REP-CONS algorithm (see Figure 1) has three main goals: to have all correct nodes agree on the same vector of input values *vals*, to agree on $Output_p$ and to have $Output_p$ change as little as possible. The first goal is achieved by executing *Byzantine* agreements on each node q's input value (Line 2.b) and by storing the result in $vals[q]$ (Line 3). This ensures that, all correct nodes have a vector, *vals*, of agreed input values.

The second and third goals are achieved by the update rule in Line 2.a; however, this update is dependent on the previous value of $Output_p$. Thus, all correct nodes are also required to agree on the previous value of $Output_p$; which is done by executing a *Byzantine* consensus BYZ-CONS (Line 2.c). The feedback update rule used in Line 2.a (specifically, the second line) is essential to the stabilizing nature of SS-REP-CONS. If there was a "static" update rule instead (one which ignores the previous value of $Output_p$) it would incur more output changes than required by the present value.

To facilitate the above method of operation, the BYZ-CONS instance and the different BYZ-AGREE$_p$ instances must start their execution at all correct nodes at the same round, and be executed properly by all correct nodes (BYZ-CONS, and BYZ-AGREE$_p$ are not self-stabilizing). Thus, a self-stabilizing pulsing algorithm $\mathcal{P}$ is used as a building block[4].

$\mathcal{P}$'s convergence time is $\Delta_{\mathcal{P}}$ and $\mathcal{P}$'s *Cycle* is set to be long enough to execute BYZ-CONS and BYZ-AGREE between two consecutive pulses. Once $\mathcal{P}$ stabilizes, each time it pulses the correct nodes start executing a new instance of BYZ-CONS and the different BYZ-AGREE$_p$s. These algorithms terminate before the next pulse of $\mathcal{P}$ and thus all correct nodes have an agreed view of *vals* and of $Output_p$. From this point on, all correct nodes continue to agree on *vals* and on the previous value of $Output_p$ and thus also on the new value of $Output_p$ ($Output_p$ is dependent only on the value of *vals* and on the previous value of $Output_p$).

REMARK 5.1. *In the context of* SS-REP-CONS Δ_{node} *is defined to be equal to* $\Delta_{\mathcal{P}}$.

5.1 Correctness Proof

LEMMA 1. *If the system has been coherent for* $\Delta_{\mathcal{P}}$ *beats, then for as long as the system stays coherent: all correct nodes enter Line 2 once every Cycle beats and do so in unison.*

PROOF. Follows immediately from the properties of a self-stabilizing *Byzantine* tolerant pulsing algorithm. See [5] for more information on pulsing algorithms. □

[3] For a detailed discussion on the applicability of the computational model's lower bounds, see Section 2.1.

[4] A pulsing algorithm invokes "pulses" every *Cycle* rounds at all correct nodes simultaneously (see [5]).

$\mathcal{P}$ is a self-stabilizing pulsing algorithm. Thus, after $\Delta_{\mathcal{P}}$ rounds, $\mathcal{P}$ stabilizes and starts pulsing in a regular pattern. Denote the round at which $\mathcal{P}$ has stabilized by $r_{stabilize}$. Consider r_{start} to be the first round at which $\mathcal{P}$ invokes a pulse after $r_{stabilize}$; in other words, r_{start} is the first round in which a pulse is invoked after $\mathcal{P}$ has stabilized.

LEMMA 2. *If the system is coherent for Cycle rounds after r_{start}, then for as long as the system stays coherent: all correct nodes have the same value of vals and the same value of $Output_p$.*

PROOF. Starting at round r_{start}, all correct nodes start executing BYZ-AGREE$_p$ for each node p. Since $\mathcal{P}$'s *Cycle* is long enough to allow a *Byzantine* agreement to terminate, all correct nodes terminate all the BYZ-AGREE$_p$ instances (even for a *Byzantine* node p) with agreed output values. Thus, when updating the values of the vector *vals* at Line 3, all correct nodes update it in the same manner. Since Line 3 is the only line in which *vals* is updated, we have that starting from round $r_{start} + Cycle$, all correct nodes have the same value for *vals*.

Notice that $Output_p$ is updated in two locations: Line 1.a and Line 4. For the same reason as in the above paragraph, all correct nodes update $Output_p$ in the same manner in Line 4. In addition, since all correct nodes have the same view of *vals* and $Output_p$ (starting from round $r_{start} + Cycle$), they all update $Output_p$ in the same way also in Line 1.a. And we have that starting from round $r_{start} + Cycle$, all correct nodes have the same view of *vals* and of $Output_p$. □

Lemma 2 implies that the "*agreement*" property holds from round $r_{start} + Cycle$ and onwards. This is because the output O_p of node p is determined by Line 5 - which sets $O_p := Output_p$ - and from the above lemma all correct nodes have the same value of $Output_p = O_p$.

Let r_{start} be as defined above, and let $r_{end}, r_{end} \geq r_{start} + 3 \cdot Cycle$ be any round such that the system is coherent in the interval $[r_{start}, r_{end}]$.

LEMMA 3. *"Validity" of the $(k, 2 \cdot Cycle)$-repetitive consensus behavior holds for* SS-REP-CONS *during the interval $[r_{start} + Cycle, r_{end}]$ (for any k).*

PROOF. According to the previous lemma, all correct nodes have the same view of *vals* in the interval $[r_{start}+Cycle, r_{end}]$. Let $[r_1, r_2] \subset [r_{start} + Cycle, r_{end}]$ be the round interval in which all non-faulty nodes have stable input values. Within *Cycle* rounds of r_1, a pulse will be invoked by $\mathcal{P}$, causing all non-faulty nodes to start executing BYZ-AGREE$_p$; all the BYZ-AGREE$_p$ instances terminate before the next pulse is invoked (no later then $r_1 + 2 \cdot Cycle$). Thus, during the interval $[r_1 + 2 \cdot Cycle, r_2]$ all correct nodes have the same view of *vals*, which reflects the "real" input values of each correct node. Since correct nodes enforce $Output_p$'s adherence to the validity requirement (see Line 2.a), then during the interval $[r_1 + 2 \cdot Cycle, r_2]$ the output of SS-REP-CONS conforms to *validity*. □

LEMMA 4. *The "Bounded changes" property of the $(2, 2 \cdot Cycle)$-repetitive consensus behavior holds for* SS-REP-CONS *during the interval $T = [r_{start} + Cycle, r_{end}]$.*

PROOF. Let $[r_1, r_2] \subset T$ (where $r_2 - r_1 \geq 2 \cdot Cycle$) be an interval in which the inputs are stable. Recall that all nodes see the same value of *vals*, and that *vals* reflects the input values of the correct nodes. Let C_v be the number of non-faulty nodes with input value v. Clearly, $\sum_v C_v = n - f$. Assume by contradiction that two different $C_v, C_{v'} \geq n-2f$; therefore $C_v + C_{v'} \geq 2n - 4f$; and since $3f < n$ we have that $C_v + C_{v'} > n - f$. But this contradicts the fact that $C_v + C_{v'} \leq \sum_v C_v = n - f$. Thus, $C_v \geq n - 2f$ holds for at most one C_v.

First, consider the case in which no C_v is $\geq n - 2f$. In this case, when executing Line 2.a, the value of $Output_p$ is either unchanged or changes to $\perp$. That is, it changes at most once.

Now consider the case in which some C_v is $\geq n - 2f$. In this case, when executing Line 2.a, the value of $Output_p$ is either unchanged, v, or $\perp$. In addition, notice that once $Output_p = v$ it cannot change to some other value unless a correct node has changed its value. Thus, $Output_p$ changes at most twice (to $\perp$ and then to v). □

THEOREM 4. SS-REP-CONS *solves the 2-SSRC problem, for $\Delta := 2 \cdot Cycle$ (with $\Delta_{stabilize} := \Delta_{\mathcal{P}} + 4 \cdot Cycle$).*

PROOF. From Lemma 1, Lemma 2, Lemma 3 and Lemma 4 we have that if the system has been coherent for a period of $\Delta_{\mathcal{P}} + 4 \cdot Cycle$, then the following holds:

1. $\mathcal{P}$ "stabilizes" after Δ_p rounds; denote by $r_{stabilize}$ the round at which $\mathcal{P}$ "stabilizes".
2. $\mathcal{P}$ will invoke a pulse at some round in the interval $[r_{stabilize}, r_{stabilize}+Cycle]$; let r_{start} denote this round.
3. Let r_{end} be the maximal round such that during the interval $[r_{start} + Cycle, r_{end}]$ no transient faults occur. The properties of the $(2, 2 \cdot Cycle)$-repetitive consensus behavior hold during $[r_{start} + Cycle, r_{end}]$.

From the above, SS-REP-CONS solves the 2-SSRC problem, for $\Delta := 2 \cdot Cycle$, and $\Delta_{stabilize} := \Delta_{\mathcal{P}} + 4 \cdot Cycle$. □

In the binary setting (the input value range contains 2 values) SS-REP-CONS matches the lower bound.

THEOREM 5. SS-REP-CONS *solves the 1-SSRC problem when $|V| = 2$.*

PROOF. The proof is the same as above, with the following observation: when $|V| = 2$, the output can change only once.

We can thus follow the lines of the proof of Lemma 4, with a single change: in the case where some $C_v \geq n - 2f$, executing Line 2.a can change $Output_p$ only to v: if $Output_p = v$ then $Output_p$ never changes. If $Output_p = 1 - v$, then it cannot change to $\perp$, since if $Output_p$ does not appear $f + 1$ times in *vals* then $1 - Output_p = v$ appears $n - f$ times in *vals*. Thus, $Output_p$ is changed to v.

With the above modification, Lemma 4 proves that when $|V| = 2$ there is at most a single change in output. Thus, together with the rest of the lemmata, SS-REP-CONS solves the 1-SSRC problem. □

6. RANGE VALIDITY

In the above sections, the "exact value" (EV for short) *validity* was discussed. It is interesting to consider a different *validity* requirement: the "range value" validity (RV for

short)[5], which states that the output of the function $\mathcal{F}$ is in the range of input values of correct nodes. Notice that EV may output $\perp$, while RV must always output a value $v \in V$. Formally, RV is defined as follows: if $\mathcal{F}(\vec{x}) = \nu$ then $\sum_{\nu' \leq \nu} \#\nu'(\vec{x}) \geq f+1$ and $\sum_{\nu' \geq \nu} \#\nu'(\vec{x}) \geq f+1$.

In the next subsections, we show that 0-SSRC cannot be solved for RV-*validity*, for any value of $f > 0$. In addition, we also show how to solve the 1-SSRC problem for RV-*validity*, when $n > 4f$, thus matching the lower bound. Table 2 presents the lower and upper bounds regarding RV-*validity*.

Table 2: Summary of results for RV-*validity*

parameters	count-instability	theorem#
$f > 0$	≥ 1	Theorem 6
$v \geq n, n + \lfloor \frac{n-1}{2} \rfloor < 5f$	≥ 2	Theorem 7
$n \geq 4f+1$	1	Theorem 8

We start with the impossibility results followed by upper bounds.

6.1 Impossibility Results: Lower Bounds

Both Theorem 1 and Theorem 2 hold for RV-*validity*, as Theorem 6 states. Theorem 7 extends the bounds for RV-*validity* only.

THEOREM 6. *For any RV system S with $f \geq 1$ the count-instability of S is at least 1, and the length-instability of S is ∞.*

PROOF. Same proof as for EV-*validity*. □

THEOREM 7. *For any RV system S with $v \geq n$ and $n + \lfloor \frac{n-1}{2} \rfloor < 5f$ the count-instability of S is at least 2.*

PROOF. Let m_0 be any state and $\vec{x_0} = (0, 1, \ldots, n-1)$. Due to RV-validity, $f \leq \mathcal{F}(\vec{x}_0, m_0) \leq n - f - 1$. There are 2 cases: $\mathcal{F}(\vec{x}_0, m_0) \leq \lfloor \frac{n-1}{2} \rfloor$ and $\mathcal{F}(\vec{x}_0, m_0) > \lfloor \frac{n-1}{2} \rfloor$; for symmetry reasons they are equivalent.

We can thus consider only the case $\mathcal{F}(\vec{x}_0, m_0) \leq \lfloor \frac{n-1}{2} \rfloor$. Mark by $k := \mathcal{F}(\vec{x}_0, m_0)$, thus $f \leq k \leq \lfloor \frac{n-1}{2} \rfloor < 2f$. Let

$$\vec{x}_1 = (\underbrace{n, n, \ldots, n}_{k-f+1}, \underbrace{k-f+1, k-f+2, \ldots, n-1}_{n-k+f-1}) .$$

Consider $\mathcal{F}(\vec{x}_1, m_1)$, where $m_1 := \tau(\vec{x}_0, m_0)$. Notice that $\vec{x}_1$ contains $k-f+1$ values of "n", thus (due to RV-validity) $f - (k-f+1) = 2f-k-1$ additional values are "to be ignored". Therefor, the output must be $\leq n - 1 - (2f - k - 1) = n+k-2f$. That is, due to RV-validity we have that $k+1 \leq \mathcal{F}(\vec{x}_1, m_1) \leq n+k-2f$. Since $k = \mathcal{F}(\vec{x}_0, m_0)$ it holds that $\mathcal{F}(\vec{x}_0, m_0) \neq \mathcal{F}(\vec{x}_1, m_1)$.

Define $k' = \mathcal{F}(\vec{x}_1, m_1)$. If $k' < 2f$ then define

$$\vec{x}_2 := (\underbrace{n, n, \ldots, n}_{f}, \underbrace{f, f+1, \ldots, n-1}_{n-f}) .$$

In this case, $2f \leq \mathcal{F}(\vec{x}_2, m_2) \leq n-1$ (where $m_2 := \tau(\vec{x}_1, m_1)$). And we have that $\mathcal{F}(\vec{x}_1, m_1) \neq \mathcal{F}(\vec{x}_2, m_2)$. Therefore, the run from m_0 on the path $\vec{x}_0, \vec{x}_1, \vec{x}_2$ has 2 output changes.

On the other hand, consider the case where $k' \geq 2f$. Let

$$\vec{x}_2 := (\underbrace{0, .., 0}_{k-f+1}, \underbrace{k-f+1, k-f+2, .., n+k-2f}_{n-f}, \underbrace{0, .., 0}_{2f-k-1}) .$$

[5]RV's definition is taken from [7]. Note that the current paper's EV definition differs from the EV in [7].

Notice that $\vec{x}_2$ contains exactly f "0"s. Moreover, $\vec{x}_2$ contains exactly f values that are $> n+k-3f$; thus, due to RV-validity we have that $\mathcal{F}(\vec{x}_2, m_2) \leq n+k-3f$. Since $k \leq \lfloor \frac{n-1}{2} \rfloor$, it holds that that $\mathcal{F}(\vec{x}_2, m_2) \leq n + \lfloor \frac{n-1}{2} \rfloor - 3f$.

We assumed $k' \geq 2f$ and since $n + \lfloor \frac{n-1}{2} \rfloor < 5f$, thus $\mathcal{F}(\vec{x}_2, m_2) < k' = \mathcal{F}(\vec{x}_1, m_1)$. That is, there were 2 output changes. □

6.2 Solving 1-SSRC for RV-validity

6.2.1 Definitions

Prior to solving the 1-SSRC problem, a definition of the problem in the context of RV-*validity* is required. The only change in the definition from EV-*validity* is in the "Validity" property of the (k, Δ)-repetitive consensus behavior definition. We thus adapt this definition in the following manner:

DEFINITION 6.1. *For RV-*validity *let the "Validity" property of the (k, Δ)-repetitive consensus behavior be:*
For any beat $r \in R'_{\lfloor \Delta}$, if $\mathcal{V}^r$ is the output value then some non-faulty node has input value $\nu_1 \leq \mathcal{V}^r$ and some non-faulty node has input value $\nu_2 \geq \mathcal{V}^r$.

COROLLARY 2. *If $v \geq n$ and $n + \lfloor \frac{n-1}{2} \rfloor < 5f$, the 1-RC (and 1-SSRC) problem cannot be solved for RV-validity.*

PROOF. Follows immediately from Theorem 7. □

6.2.2 Algorithm

The algorithm SS-REP-CONS also solves the 1-SSRC problem - for $n \geq 4f+1$ - in the RV-*validity* setting, provided that we replace Line 2.a with the following:

- **Remove** f lowest and f highest values from *vals*. Let *high* be the new highest value in *vals*, and *low* be the new lowest value (after the removal);
- **If** $Output_p \notin [low, high]$ then set $Output_p$ to be the median value in *vals*.

This change incorporates the difference between EV-*validity* and RV-*validity*.

6.2.3 Proofs

THEOREM 8. SS-REP-CONS *(with the above changes) solves the $1-SSRC$ problem for $n \geq 4f+1$.*

PROOF. Notice that Lemma 1, Lemma 2 and Lemma 3 all hold for RV-*validity*. Thus, it is left to show that once the system has stabilized, it does not change its output more than once. Consider v' to be the value of $Output_p$ at all nodes after SS-REP-CONS has stabilized (v' is well defined due to Lemma 2).

Consider the first beat in which $Output_p$ changes to $v \neq v'$; Sort the input values of the correct nodes, and let v_{low} be the $f+1$st input value from the bottom, and v_{high} be the $f+1$st from the top. The median value v has at least $2f+1$ values that are lower or equal to it. There are at most f faulty nodes, and thus there are at least $f+1$ correct nodes lower or equal to v. Thus, $v_{low} \leq v$. For the same reason $v_{high} \geq v$. Thus, the calculated median is in the range $[v_{low}, v_{high}]$; in other words, $v \in [v_{low}, v_{high}]$.

In any future beat, when values are removed from *vals*, the values v_{low} and v_{high} are not discarded, since only the f

lowest and f highest values are removed. Thus, when calculating low and $high$ (see algorithm in previous subsection) the following holds: $low \leq v_{low}$ and $high \geq v_{high}$. Thus, $Output_p \in [low, high]$, which means that $Output_p$ is not updated; *i.e.* , $Output_p = v$. Thus, the output value of the correct nodes changes at most once after SS-REP-CONS has converged. □

7. DISCUSSION

7.1 Related Problems

There are two problems that seem to be related to repetitive consensus: self-stabilizing *Byzantine* agreement, and continuous consensus. However, both problems differ from repetitive consensus on the same major issue.

Self-stabilizing *Byzantine* Agreement (SSBA): This problem consists of having a *Byzantine* agreement that is self-stabilizing (see [3] for more information). That is, starting from an arbitrary memory state, any node p can initiate an agreement procedure on its value. Due to the self-stabilizing property, p can repeatedly initiate agreement procedures on its value; thus, in a sense, SSBA can be seen as a repetitive *Byzantine* agreement. The main difference between repetitive consensus and SSBA lies in the difference in their *validity* property. SSBA requires that if the leader is non-faulty, then all non-faulty nodes agree on the leader's value. Since SSBA requires an agreement on a single (leader) node's value, if this value changes then the agreement should change with it. However, in repetitive consensus, we require a consensus of all nodes' values: thus, if one node's value changes, and this node is *Byzantine*, then the output value should not change. Therefore, in repetitive consensus, a single node changing its input value must not lead to a change in the output, where as in SSBA a single node's change of input value should lead (if it is the leader) to the change of the output value.

Continuous Consensus (CC): The CC problem involves continuously agreeing on all the inputs of all the nodes in the system. That is, each node should have a list of all the inputs that each node in the system has had until now (see [12] for more information). However, the property that all nodes agree on the input values of all other nodes during the entire execution is still not sufficient to solve the 0-RC problem.

As in SSBA, the essence of this impossibility lies in the requirement that if a *Byzantine* node changes its input value, the output value of the repetitive consensus does not change. Thus, in CC, even if all non-faulty nodes have all the input values of all nodes, they cannot differentiate between a non-faulty node that has changed its value, and a *Byzantine* node mimicking a new input value.

REMARK 7.1. *The above comparison leads to the conclusion that the impossibility result of Corollary 1 stems from the requirements of the repetitive consensus problem, and not from the ability to gather information from the entire system. In other words, it is not missing data that prohibits the repetitive consensus output; there simply does not exist a function on the nodes' inputs that satisfies the requirements of the 0-RC problem.*

7.2 Pseudo Self-stabilization

In the context of self-stabilization, algorithms that converge to a "safe" state and leave the safe states at most a constant number of times are called "pseudo self-stabilizing" algorithms (see [6]). Depending on how one defines a *safe configuration* in the context of *Byzantine* faults, the algorithms presented in the current paper may be considered as pseudo self-stabilizing, where the constant number of times they may leave "safe" states is 1 or 2.

7.3 Bounded-delay Network

In the full paper the results herein are extended to the bounded-delay model, in which there is no global-beat-system; instead, a bound on messages' delivery time is assumed. The main idea behind this extension is the usage of a "transformer" from the global-beat-system model to the bounded-delay model, which conserves the properties of SS-REP-CONS.

7.4 Open Questions

We have shown that for any system with $f > 0$ the count-instability is at least 1, and that this is tight for $|V| = 2$. In addition, for systems with $n = 3f + 1$ (where $|V| \geq 3$) we have shown that the count-instability is at least 2, and SS-REP-CONS reaches this bound. This raises the following question: what is the exact relation between $|V|, n$ and f regarding count-instability and what algorithm can achieve it?

When considering the RV-*validity* scenario it becomes even more interesting: we have shown how to solve the 1-SSRC problem for any $n \geq 4f + 1$. What happens for smaller n? For $n \leq 4f$ we have only given lower bounds. Can upper bounds be given for $n \leq 4f$?

Acknowledgements

We would like to thank Yoni Peleg for contributing Theorem 5.

8. REFERENCES

[1] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, March 2004.

[2] A. Bar-Noy, X .Deng, J. Garay, and T. Kameda. Optimal amortized distributed consensus. *Information and Computation*, 120(1):93–100, 1995.

[3] A. Daliot and D. Dolev. Self-stabilizing byzantine agreement. In *Proc. of the Twenty-fifth ACM Symposium on Principles of Distributed Computing (PODC'06)*, Denver, Colorado, Jul 2006.

[4] L. Davidovitch, S. Dolev, and S. Rajsbaum. Stability of multivalued continuous consensus. *SIAM Journal on Computing*, 37(4):1057–1076, 2007.

[5] D. Dolev and E. N. Hoch. On self-stabilizing synchronous actions despite byzantine attacks. In *Proc. the 21st Int. Symposium on Distributed Computing (DISC'07)*, Lemesos, Cyprus, Sep. 2007.

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

[7] S. Dolev and S. Rajsbaum. Stability of long-lived consensus. *J. Comput. Syst. Sci.*, 67(1):26–45, 2003.

[8] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In Marek Karpinski, editor, *FCT*, volume 158 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 1983.

[9] Eli Gafni, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. The committee decision problem. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 502–514. Springer, 2006.

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

[11] Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Trans. Comput. Syst.*, 8(4):284–304, 1990.

[12] Tal Mizrahi and Yoram Moses. Continuous consensus via common knowledge. In *TARK '05: Proceedings of the 10th conference on Theoretical aspects of rationality and knowledge*, pages 236–252, Singapore, Singapore, 2005. National University of Singapore.

[13] Roberto De Prisco, Butler Lampson, and Nancy Lynch. Revisiting the PAXOS algorithm. *Theoretical Computer Science*, 243(1–2):35–91, 2000.

[14] Marco Schneider. Self-stabilization. *ACM Comput. Surv.*, 25(1):45–67, 1993.

[15] F. Becker and S. Rajsbaum and I. Rapaport and E. Re'mila. Average binary long-lived Consensus: quantifying the stabilization role played by memory. In *Proc. 15th International Colloquium on Structural Information and Communication Complexity (SIROCCO'08)*, Switzerland, June. 2008.