Asynchronous and Fully Self-Stabilizing Time-Adaptive Majority Consensus

Janna Burman[1], Ted Herman[2], Shay Kutten[1⋆], and Boaz Patt-Shamir[3]

[1] Dept. of Industrial Engineering & Management
Technion, Haifa 32000, Israel.
bjanna@tx.technion.ac.il, kutten@ie.technion.ac.il

[2] Dept. of Computer Science
University of Iowa, Iowa City, Iowa 52242, USA.
herman@cs.uiowa.edu

[3] Dept. of Electrical Engineering
Tel-Aviv University, Tel Aviv 69978, Israel.
boaz@eng.tau.ac.il

Abstract. We study the scenario where a batch of transient faults hits an asynchronous distributed system by corrupting the state of some f nodes. We concentrate on the basic *majority consensus* problem, where nodes are required to agree on a common output value which is the input value of the majority of them. We give a fully self-stabilizing adaptive algorithm, i.e., the output value stabilizes in $O(f)$ time at all nodes, for any unknown f. Moreover, a state stabilization occurs in time proportional to the (unknown) diameter of the network. Both upper bounds match known lower bounds to within a constant factor. Previous results (stated for a slightly less general problem called "persistent bit") assumed the synchronous network model, and that $f < n/2$.

1 Introduction

We consider protocols that can withstand *state-corrupting faults* that flip the bits of the volatile memory in a system arbitrarily. A system that reaches a legitimate state starting from an arbitrary state is called *self-stabilizing* [14] or *fully self-stabilizing.*[4] The *stabilization time* is the time that elapses since the protocol starts executing (with arbitrary states at the corrupted nodes) until the system reaches a legal state. Classical self-stabilizing protocols were designed to minimize worst-case stabilization time regardless of the number of nodes whose state was corrupted by the faults. More recently, it has been recognized that if the faults hit only a few nodes, then a much faster stabilization is possible, see e.g. [25, 2, 29, 30]. In [29, 30] a system is called *time-adaptive* or *fault-local*

⋆ This research was partially supported by the Israel Science Foundation (Grant 4005/02)

[4] We use the qualifier "fully" to emphasize that the state can be arbitrarily corrupted. We mention some weaker forms of stabilization later.

if its stabilization time is proportional to the number of nodes whose state was corrupted.

The *Majority Consensus* problem is a basic problem in distributed computing: each node has an input, and it is required that the output at each node stabilizes to the majority of these inputs. In this paper it is assumed that the input can be changed by transient faults, by the environment, or by the stabilizing algorithm. This problem is a simple form of the general consensus problem [20], which is fundamental to fault tolerant distributed applications.

Our results. We present a fully self-stabilizing, optimal time-adaptive solution for the majority consensus problem for asynchronous networks. The output of our algorithm stabilizes in time proportional to f, the number of nodes hit by faults. The state stabilization time is proportional to the network diameter. In other words, our algorithm is optimal in both output and state stabilization (see [29]). These properties hold even in the case that $f \geq n/2$ (where n is the number of nodes). This should be contrasted with previous results that were only for $f < n/2$.

As a corollary, our solution solves the *Persistent Bit* problem [30, 29] whenever such a solution is possible. "Persistent Bit" is the task of remembering the value of a replicated bit in the face of state corruptions. The time adaptive solution in [29] was only for *synchronous* networks, while we solve it time adaptively for asynchronous networks.

The algorithm utilizes some known techniques, namely the self-stabilizing synchronization [6] and the power-supply method [1]. We use a new version of the power-supply method. The time-adaptivity we prove for power-supply is stronger than the self-stabilization proven in [1]; this new property may be useful for other applications of power supply.

For simplicity, we present the algorithm as one maintaining only 0/1 values, but it can easily be adapted for any range of values.

Related Work. The study of self-stabilizing protocols was initiated by Dijkstra [14]. *Reset-based* approaches to self-stabilization are described in [27, 3, 7, 6, 16, 5]. One of the main drawbacks of this approach is that the detection mechanism triggers a system-wide reset in the face of the slightest inconsistency. Fast stabilization of the output variables are demonstrated in a number of algorithms [26, 24, 2, 22, 4, 11, 32, 8]. Some general methods to achieve time adaptivity are discussed in [29, 17, 21]. The distinction between output stabilization and state stabilization (see definitions in Sec. 2) is used and discussed in a number of papers [29, 31, 15, 25, 23].

We use the self stabilizing synchronizer with a counter (sometimes called *phase clock*) of [6]. Other phase clocks in the literature, such as [10, 18, 13, 23], may also be useful.

Papers most closely related to our work are [6, 25, 2, 29, 30, 22, 11, 32]. A preliminary brief announcement [28] at the PODC'98 symposium announces results that appear in the current paper.

Paper organization. In Sec. 2, we formalize the model and introduce a few notations. In Sec. 3, we present the problems and explain the overall structure of the solution. The new algorithm is presented in Sec. 4 and 5 dealing with output and state stabilization respectively.

2 Model and Notations

System Model. The system topology is represented by an undirected graph $G = (V, E)$, where nodes represent processors and edges represent communication links. The number of the nodes is denoted by $n = |V|$. The diameter of the graph is denoted by **diam**. We assume that there is a known upper bound on the diameter of the network, denoted by D. This upper bound serves only for the purpose of having finite space protocols. For $i \in V$, we define $N(i) = \{j \mid (i, j) \in E\}$, called the *neighbors* of i. We do not assume that the set of edges in the network is known in advance, i.e., algorithms are required to work on any topology. We consider an *asynchronous message passing network* model. In a message passing network, processors may exchange values only by transmitting *packet*s. In our model, a packet consists of a set of *message*s. The packet delivery time can be arbitrary, but for the purpose of time analysis only, we assume that each packet is delivered in at most one time unit.

The number of packets that may be in transit on any link in each direction and at the same time is bounded by some parameter B (independent of the network size). We adopt this assumption from [1]; it is necessary, as shown in [19], for solving problems in a self-stabilizing manner. For simplicity, we assume that $B = 1$, i.e., there is at most one outstanding packet on each link at any given time (see [7] for more details). A packet may contain any number of messages. Each message contains information (a node identity and more) related to some specific node. In our algorithm, we distinguish between two types of messages, called *strong* and *weak* (Sec. 4.1). Each node maintains two buffers for each link: one for the incoming packets and another one for the outgoing packets. Each buffer contains at most one message for each type and for each node: for each type and node, only the most recent message is stored in the buffer (cf. [1]). If a new message arrives, the previous one is discarded.

We adopt the usual definitions of the following: a local state of a node (an assignment of values to the local variables and the location counter); a global state of a system of nodes (the cross product of the local states of its constituent nodes, plus the contents of the links); the semantics of protocol actions (possible atomic steps and their associated state changes); an execution sequence of protocol P (a possibly infinite sequence of global states in which each element follows from its predecessor by execution of a single atomic step of P).

Fault Model. We follow the terminology of [29]. We assume that each protocol has a *legality predicate* over the set of global states. Informally, a legal global state of our protocol is a state in which the protocol is ready to get a new batch of faults in the sense that following these faults the protocol satisfy all its claimed execution properties (in particular, time adaptivity; as for state stabilization, our protocol stabilizes from any finite number of faults that occur anytime). A formal definition of the legality predicate of our protocol is deferred to the full paper. A *faulty node* is defined as in [6]. We define two global states: a *start state* s_0 that exists at time $t_0 = 0$, right after the faults hit, and a *reference state* s_{-1}, where s_{-1} is legal; a node is called faulty if its local state is different in s_0 and s_{-1}. Note that every part of the state (except for the ID of the node) can be changed by the adversary and this is considered a fault. Without loss of generality, a fault that corrupts a packet over a link is considered a fault in the node that receives that packet eventually. A *fault number* is the number of faulty nodes at the start state s_0.

A protocol is called *f-stabilizing* if starting from a state with fault number at most f, it reaches a legal state eventually and remains in a legal state thereafter. A protocol is called *self-stabilizing* or *fully self-stabilizing* if it is f-stabilizing for $f = n$. We distinguish between *output stabilization* and *state stabilization*: output stabilization is said to occur once the externally observable portion of the state becomes (and stays) legal, and state stabilization is said to occur when the entire state becomes (and stays) legal. The maximum number of time units that takes to reach the state stabilization (respectively, output stabilization) is called the *state stabilization time* of the protocol (resp., *output stabilization time*). If the output stabilization time of an algorithm depends only on the fault number then the algorithm is said to be *fault local*, or *time-adaptive.*

Typographical Convention. Protocol variables are represented using `teletype font`, with a subscript indicating the node in which the variable is located. For example, $\texttt{dist}_i$ denotes the "distance" variable at node i, whose value may be arbitrary. Graph properties are represented using a **boldface font**, as in $\mathbf{dist}(i, j)$, which denotes the true distance in the graph between nodes i and j.

3 The Majority Consensus Problem

A node is said to be *internally legal* if its local state can be reached in an execution with no faults.

Our main target is the following problem.

Definition 1. *In the Majority Consensus problem, each node has an* input bit *that can be changed by the environment and an externally observable* output bit*. The bits satisfy the following requirements.*

- Eventual Agreement: *eventually, all output and input bits must be equal.*
- Majority Consensus: *If there is a majority of internally legal nodes having a common value b in all the input and output bits at the start state, then the eventual common value of all the output and input bits is b.*

Let us also define the Persistent Bit problem that was dealt with previously in [30, 29]. In this problem, each node has an *input bit* that can be changed by the environment and an externally observable *output bit*. It is assumed that all input bits were assigned a common value b, and then a fault may have occurred. The "Eventual Agreement" requirement is identical to that of the Majority Consensus problem. However, the *Persistence* requirement (which replaces the "Majority Consensus" requirement) is that all output and input bits eventually stabilize to the original value b. It is not hard to see that Persistence is impossible if $f \geq n/2$. In the case of $f < n/2$, the Persistent Bit problem can be reduced to the Majority Consensus problem, since in this case there is a majority of non-faulty nodes in the start state.

In [29], an algorithm was presented for which the following result was proven.

Theorem 1. *There exists a protocol for the* ***Persistent Bit*** *problem in the synchronous network model such that if the local states of* $\underline{f < n/2}$ *of the nodes are changed arbitrarily, then the output bits are restored everywhere in* $O(f)$ *time units, and the state stabilization occurs in* $O(\mathbf{diam})$ *time units, where* **diam** *denotes the actual unknown diameter of the network.*

Here we prove the following strictly stronger result.

Theorem 2. *There exists a protocol for the* ***Majority Consensus*** *problem in the asynchronous network model such that if the local states of* $\underline{f \leq n}$ *of the nodes are changed arbitrarily, then the output bits stabilize everywhere in* $O(f)$ *time units, and the state stabilization occurs in* $O(\mathbf{diam})$ *time units, where* **diam** *denotes the actual unknown diameter of the network.*

As explained above, a solution to the majority consensus problem is a solution to the persistent bit problem. Thus, the improvement in our result is twofold: first, the algorithm presented here is for the strictly weaker model of asynchronous communication; and second, our algorithm can withstand any number of faults (i.e. it is fully self-stabilizing for the majority consensus problem).

3.1 Overview of the protocol

The high-level structure of the new protocol is identical to that of the algorithm presented in [29]. The protocol has two parts: the *output stabilization* (OS) protocol and the *input fixing* (IF) protocol. The input is given replicated at all the nodes. Then, if faults corrupt a minority of the input bits, they can be repaired by adopting the value of the majority. Here, if the majority is corrupted, the correct minority is brought to agree with the majority if a majority exists for some input value in internally legal nodes. Otherwise, the protocol chooses some legal value - the same at all the nodes. To perform this repair, the input bit of each node is disseminated to all the other nodes using a protocol called *power-supply regulated broadcast* (PS-RB) [29]. The (externally observable) output bit is computed at each node by taking the majority of the values received by PS-RB. For the input fixing part we design an algorithm that works independently from the output stabilization algorithm and stabilizes the input bits to legal values in $O(\mathbf{diam})$ time units.

While the high-level structure of the solution resembles the one in [29], component sub-protocols had to be changed. First, OS was changed because of the asynchrony. In [29], the propagation of broadcasted input values was slowed down. The idea was to allow fault detection messages to catch up with faulty broadcast messages and stop them. Slowing down is easy in a synchronous model: say, by forwarding slow messages every other clock "tick" (called *pulse*). This method cannot be used in asynchronous systems. Instead, we use the Power Supply technique [1]. This change enables the use of OS in asynchronous networks. No change to OS was needed to ensure full stabilization, i.e., to allow for the case $f \geq n/2$. This is because, even if $f \geq n/2$, the new Input Fixing makes sure that complete state stabilization occurs in $O(\mathbf{diam})$ and hence, in $O(n)$ time. Note that for $f \geq n/2$, the time adaptivity requirement is vacuous since in that case, $f = \Theta(n)$, and hence, an output stabilization time of $O(n)$ is good enough.

Second, the new IF is a fully-stabilizing protocol for any f. To design the IF algorithm, we first construct a fully-stabilizing algorithm which solves the case of $f \leq n$ for the *synchronous* network model. We then combine it with the self-stabilizing synchronizer to make it work in the asynchronous network model. Such a use of a synchronizer was not possible for OS, since known synchronizers are not time adaptive. The use of a synchronizer is possible for IF, since IF cannot be time adaptive anyway [29].

4 Output Stabilization

4.1 The Output Stabilization protocol

The main tool of the OS protocol is that each node has faithful replicas of all input values in the system. These replicas, called *estimates,* are used to compute the local output bit by a majority rule. For now, assume that for $f < n/2$, input bits at non-faulty nodes never change (we prove this later). Under this assumption, it is sufficient for time-adaptivity that (1) in $O(f)$ time all unfaithful estimates (those damaged by the faults) disappear, and (2) at each node there are at least $f+1$ (a majority) faithful estimates of non-faulty nodes. In this way, after $O(f)$ time, the majority vote at each node outputs the original value that was at the nodes at the reference state s_{-1} before the faults (Theorem 3).

In the case that $f \geq n/2$, the output stabilization is achieved in two stages: first, input bits of all nodes stabilize (by the IF protocol) in $O(\mathbf{diam})$ time (Sec. 5); second, output bits stabilize to the common value of the input bits (by the OS protocol) in $O(\mathbf{diam})$ time too (Theorem 4).

We now introduce some terms used in the following description of the OS protocol. The term *estimate* is used to describe not only the replica of some input value, but also any other broadcast piece of information (like distance or parent pointer values used by PS-RB). Given a node $k \in V$, an estimate is said to be *faithful w.r.t. k* if: (1) it is an input value estimate and its value is identical to the input value that is broadcast by the source k, or (2) it is a distance or a parent pointer estimate and its value conforms with the graph properties. An *erased estimate* means an estimate the value of which is its default value, e.g. $\perp$ is a default value for an input bit estimate (`null` and ∞ are the default values of parent pointer and distance estimates (resp.)). An *unfaithful estimate* is one that is both not faithful and not erased. The term *unfaithful message/node w.r.t* k refers to a message/node that contains an unfaithful estimate for node k.

Let us explain the mechanism of the OS protocol. As in the algorithm presented in [29], to disseminate the input values through the system in the regulated manner, OS builds a Bellman-Ford (BF) [9] minimal hop spanning tree rooted at each node, which is "regulated" by the power-supply technique as explained below. Thus, each node $r \in V$ maintains multiple (n) BF trees: one tree to broadcast its own input value and the rest $n-1$ trees for participating in broadcasts of other nodes' input values. The invocation of the algorithm that builds such a spanning tree is independent from those that build the other trees. We term this algorithm *power-supply regulated broadcast* (PS-RB). The following description of OS applies to *one* PS-RB tree, rooted at some node r.

A fault can create at some node i an unfaithful estimate of the input of r. Moreover, a careless protocol could have disseminated the unfaithful estimate to other nodes, causing them to behave as if they were faulty too. This would have rendered time adaptive stabilization impossible. To avoid that situation, the power supply technique presented in [1] is used to regulate the broadcast of the input values. That is, the OS algorithm uses two types of messages: *strong* and *weak*. Each node i sends to its neighbors a set of *weak* messages periodically. Each weak message contains node's current estimates for every other node. Weak messages are not forwarded. The goal of the exchange of weak messages is to detect faults in nodes' states as fast as possible by detecting an *inconsistency* in states of neighbors. An inconsistency (w.r.t. node r) in some node $i \neq r$ is checked by evaluating the local predicate $inconsis_{i,p}(r)$ (given in Fig. 1) whenever a message (either weak or strong) arrives from neighbor $p \in N(i)$ (Def. 4). The predicate is local in the sense that it is computed only on variables of node i, and variables of its neighbors, received by messages from them. When an inconsistency w.r.t. r is detected at node i, i initiates a *reset wave*. This is a broadcast wave that is forwarded over the subtree (for r's broadcast) rooted at i. The reset erases all the estimates of r and r's tree structure (the subtree rooted at i) as it goes. Note that a reset in r's tree does not harm the other trees in the same nodes.

Strong messages are generated originally by each PS-RB tree root r to broadcast its own input value. They are the only messages that can propagate estimates of a particular root r. A strong message of r propagates from the broadcast tree root r to the leaves. To "adopt" new estimates for r, the following must happen for node i: (a) i must receive two identical consecutive strong messages (m_1 and m_2) containing these new estimates; (b) m_1 and m_2 must arrive on the same path from r; (c) weak messages received from the same neighbor p on that path in between m_1 and m_2 must be consistent in the sense that they do not cause the local predicate $cand\_inconsis_{i,p}(r)$ (given in Fig. 1) to be true. Node i that receives a candidate estimate (in a strong massage) for the first time, does not propagates this estimate. Instead, i "consumes" that strong message and initiates a reset wave down the tree. Only on the second receive of the same candidate estimate, node i can "adopt" and propagate this estimate. Note that new estimate "adoption" can occur only if the explained above Constrains (a)-(b) holds true.

The described mechanism ensures that unfaithful strong messages eventually disappear from the network, since: (a) strong messages cannot flow in a cycle (Obs. 2 [1]), (b) although nodes can forward unfaithful strong messages, no node can generate such messages, and (c) the number of unfaithful strong messages is reduced by each node that "consumes" it. This, in turn, prevents unfaithful strong messages from propagating unfaithful estimates too far. Thus, a reset wave eliminates the effect of unfaithful estimates on the majority function as fast as possible (in $O(f)$ time). Meanwhile, in $O(f)$ time too (for $f < n/2$), a majority (at least $f + 1$) of faithful (and correct) input value estimates of non-faulty nodes arrive (by the broadcast of faithful strong messages) and are "adopted" by each node i. Now, the majority function outputs a legal value at each i.

Pseudo-code for the output stabilization is presented in Fig. 2. Definitions for the pseudo-code are presented separately in Fig. 1. For every pair of nodes i, j, $\texttt{val}_i[j]$ is the current estimate of node i for the input value of node j, and $\texttt{val}_i[i]$ is the input value of node i. The majority function ignores the $\perp$ values and outputs 0 in the case of a tie. Variable $\texttt{dist}_i[j]$ is the current estimate of i for the shortest distance from i to j. Variable $\texttt{par}_i[j]$ is the current estimate of i for the parent pointer to the neighbor leading to j on the shortest path. Variables with the prefix `cand` are used to store candidate values for newly arrived estimates in strong messages. $\texttt{Strong}_{i,p}(j)$ and $\texttt{Weak}_{i,p}(j)$ are strong and weak messages received at node i from neighbor p and contain estimates for node j. Each message contains three elements: an identity of a node j, an estimate for the input value of node j and an estimate for the shortest distance between p and j.

4.2 Analysis of the Output Stabilization protocol

To analyze the OS part of the new algorithm we use the structure of the analysis used for the synchronous algorithm in [29]. To benefit from the work that was already performed, we conform to definitions, notations and the proof sequence as much as possible while emphasizing the differences. First, we concentrate on proofs of the output stabilization for the case of $f < n/2$.

Since the regulated broadcast on any BF tree in the system works independently of the others, we consider a single representative tree rooted at a non-faulty node j. For the case of $f < n/2$, trees rooted at faulty nodes are ignored, since they can distribute an arbitrary value.

Definition 2. *Let $i \in V$. The* depth *of i is* $\mathbf{depth}(i) \overset{\mathrm{def}}{=} \max\{\mathbf{dist}(i, j) \mid j \in V\}$.

Definition 3. *Let $j \in V$, and fix a global state.*

- *A node i is* faithful with respect to $j \neq i$ *if* $(\mathtt{val}_i[j] = \mathtt{val}_j[j]) \wedge (\mathtt{dist}_i[j] = \mathbf{dist}(i,j))$ *and there exist path of nodes* $(x_1 = j, x_2, ..., x_l, i)$*, such that* $x_l = \mathtt{par}_i[j]$ *and the length of the path is* **dist***(i,j).*
- *A node i is* faithful w.r.t. itself *if* $\mathtt{dist}_i[i] = 0$ *and* $\mathtt{par}_i[i] = \mathtt{null}$*.*
- *A strong or a weak message* $(j, value, dist)$ *is* faithful w.r.t. j *if the following condition holds:* $(value = \mathtt{val}_j[j]) \wedge (dist = \mathbf{dist}(i,j))$*.*

Definition 4. *Let $j, i \in V$ such that $i \neq j$ and fix a global state.*

- *Let $p \in \mathrm{N}(i)$. Node i is* inconsistent with p with respect to j *if Predicate* $inconsis_{i,p}(j, \mathtt{val}_p[j], \mathtt{dist}_p[j])$*, given in Fig. 1 holds true.*
- *A node i is* inconsistent w.r.t. j *if for some $p \in \mathrm{N}(i)$ $inconsis_{i,p}(j)$ holds.*

Note that there is a subtle, but important, difference between the definition of the inconsistency between nodes i and p (Def. 4) and the definition of Predicate

Constants

V	:	the set of nodes
D	:	an upper bound on **diam**
$N(i)$	:	the set of neighbors of i

State for node i

(* local estimates and candidates for the local estimates *)

$\mathtt{val}_i[V]$, $\mathtt{cand\_val}_i[V]$	:	array of $\{0, 1, \perp\}$, except for $\mathtt{val}_i[i]$ that is in $\{0, 1\}$
$\mathtt{par}_i[V]$, $\mathtt{cand\_par}_i[V]$	:	array of $N(i) \cup \{\mathtt{null}\}$
$\mathtt{dist}_i[V]$, $\mathtt{cand\_dist}_i[V]$	:	array of $\{1, \ldots, D\} \cup \{\infty\}$
$\mathtt{output}_i$	$\in$	$\{0, 1\}$

Messages at node i (* received from $p \in N(i)$ with estimates for node v *)

$\mathtt{Weak}_{i,p}(v)$, $\mathtt{Strong}_{i,p}(v) \in \{ [V, \ \{0, 1, \perp\}, \ \{1, \ldots, D\} \cup \{\infty\}] \}$

Shorthand (* *value* and *dist* are estimates for node j received from $p \in N(i)$ *)

$$
\begin{aligned}
&inconsis_{i,p}(j, value, dist) \equiv inconsis_{i,p}(j) \equiv \\
&\equiv [i \neq j] \ \wedge \Big[(value \neq \mathtt{val}_i[j] \ \wedge \ p = \mathtt{par}_i[j]) \vee \\
&\qquad (dist + 1 < \mathtt{dist}_i[j]) \vee \\
&\qquad (dist + 1 \neq \mathtt{dist}_i[j] \ \wedge \ p = \mathtt{par}_i[j]) \vee \\
&\qquad (\mathtt{par}_i[j] = \mathtt{null} \ \wedge \ \mathtt{dist}_i[j] \neq \infty) \vee \\
&\qquad (\mathtt{par}_i[j] \neq \mathtt{null} \ \wedge \ \mathtt{dist}_i[j] = \infty) \vee \\
&\qquad (\mathtt{par}_i[j] = \mathtt{null} \ \wedge \ \mathtt{dist}_i[j] = \infty \ \wedge \ \mathtt{val}_i[j] \neq \perp) \ \vee \mathtt{par}_i[j] \notin N(i)\Big]
\end{aligned}
$$

$cand\_inconsis_{i,p}(j, value, dist) \equiv cand\_inconsis_{i,p}(j) \equiv$

(* obtained by applying the $inconsis_{i,p}(j)$ on node i variables with prefix `cand` *)

$$
\begin{aligned}
&is\_candidate_{i,p}(j, value, dist) \equiv (\mathtt{cand\_val}_i[j] = value \neq \perp) \ \wedge \\
&\qquad (\mathtt{cand\_dist}_i[j] = dist + 1 \ \wedge dist \neq \infty) \ \wedge (\mathtt{cand\_par}_i[j] = p)
\end{aligned}
$$

Fig. 1. *Definitions at node i.*

```
Procedure send_weak()                                  (* sending a set of weak messages *)
    for  each $j \in V$  do
        Send [ $j, \mathtt{val}_i[j], \mathtt{dist}_i[j]$ ] as a weak message to $N(i)$

Upon receiving $\mathtt{Strong}_{i,p}(v) \equiv (v,\ msg\_value,\ msg\_dist)$ message:
                                                  (* the following is executed atomically *)
    $\mathtt{par}_i[i] \leftarrow \mathtt{null},\ \mathtt{dist}_i[i] \leftarrow 0$                        (* $i$ is the root of its tree *)
    if $inconsis_{i,p}(\mathtt{Strong}_{i,p}(v))$ then
        if $is\_candidate_{i,p}(\mathtt{Strong}_{i,p}(v))$ then
            $\mathtt{val}_i[v] \leftarrow msg\_value$
            $\mathtt{par}_i[v] \leftarrow p$
            $\mathtt{dist}_i[v] \leftarrow msg\_dist + 1$
            Send  [ $v, \mathtt{val}_i[v], \mathtt{dist}_i[v]$ ] as a strong message to $N(i)$
        else                                                  (* new information received *)
            $\mathtt{cand\_val}_i[v] \leftarrow msg\_value$
            $\mathtt{cand\_par}_i[v] \leftarrow p$
            $\mathtt{cand\_dist}_i[v] \leftarrow msg\_dist + 1$
            $\mathtt{val}_i[v] \leftarrow \perp$                            (* generate reset on inconsistency *)
            $\mathtt{par}_i[v] \leftarrow \mathtt{null}$
            $\mathtt{dist}_i[v] \leftarrow \infty$
            Send  [ $v, \mathtt{val}_i[v], \mathtt{dist}_i[v]$ ] as a weak message to $N(i)$
    else                                                    (* if consistent, just forward *)
        if  $p = \mathtt{par}_i[v]$ then
            Send  [ $v,\ \mathtt{val}_i[v], \mathtt{dist}_i[v]$ ] as a strong message to $N(i)$

    $\mathtt{output}_i \leftarrow$ majority $\{\mathtt{val}_i[j] \mid j \in V\}$

Upon receiving $\mathtt{Weak}_{i,p}(v) \equiv (v,\ msg\_value,\ msg\_dist)$ message:
                                                  (* the following is executed atomically *)
    $\mathtt{par}_i[i] \leftarrow \mathtt{null},\ \mathtt{dist}_i[i] \leftarrow 0$                        (* $i$ is the root of its tree *)
    if $inconsis_{i,p}(\mathtt{Weak}_{i,p}(v))$ then                (* generate reset on inconsistency *)
        $\mathtt{val}_i[v] \leftarrow \perp$
        $\mathtt{par}_i[v] \leftarrow \mathtt{null}$
        $\mathtt{dist}_i[v] \leftarrow \infty$
        Send  [ $v, \mathtt{val}_i[v], \mathtt{dist}_i[v]$ ] as a weak message to $N(i)$
    if $cand\_inconsis_{i,p}(\mathtt{Weak}_{i,p}(v))$ then
        $\mathtt{cand\_val}_i[v] \leftarrow \perp$
        $\mathtt{cand\_par}_i[v] \leftarrow \mathtt{null}$
        $\mathtt{cand\_dist}_i[v] \leftarrow \infty$

Do forever:                            (* each iteration of the loop executes atomically *)
    $\mathtt{par}_i[i] \leftarrow \mathtt{null},\ \mathtt{dist}_i[i] \leftarrow 0$                        (* $i$ is the root of its tree *)
    Send  [ $i, \mathtt{val}_i[i], \mathtt{dist}_i[i]$ ] as a strong message to $N(i)$
    send_weak()
```

Fig. 2. *Code for output stabilization at node i.*

$inconsis_{i,p}(j)$. The predicate is computed by the algorithm, hence it uses the variables of i and the values of the message received from p, in the buffer of i. On the other hand, Def. 4 applies to the variables of i versus the variables of p. The algorithm at i cannot access the variables of p, so it cannot know immediately whether i and p are inconsistent. However, in at most an additional (fault-free) time unit, an additional weak message that is originated in p arrives at i and the true inconsistency may be detected.

We ignore the case of inconsistency w.r.t. the node itself, since it is easy to see that the algorithm ensures a permanent consistency in this case (see the first two actions in the *Do forever* loop and in the procedures dealing with the receive of weak and strong messages in Fig. 2).

The importance of the following property of PS-RB is that it holds even before stabilization. A similar lemma was used also in [29] for the synchronous algorithm. The proof is deferred to the full paper.

Lemma 1. *Let $i, j \in V$, and let $t \geq t_0 (= 0)$. Assume that no faults occur in the time interval $[0, t]$. Then, at time $t + 1$, $\texttt{dist}_i[j] \geq \min(t, \mathbf{dist}(i, j))$.*

The following lemma implies that faithful estimates of input values that do not change, are established quickly.

Lemma 2. *Let $i, j \in V$, and let $t \geq 0$. If $\texttt{val}_j[j]$ does not change in a (fault-free) time interval $[0, 3t+3]$, then for every node i with $\mathbf{dist}(i, j) \leq t$, i is faithful w.r.t j at time $3t + 3$.*

Proof Sketch: By induction on t. The basis for $t = 0$ is trivial. For the induction step we assume that the lemma holds for some $t = k$. We now prove the lemma for $t = k + 1$. Let x_{k+1} be a node, such that $\mathbf{dist}(x_{k+1}, j) = k + 1$.

Starting at a time (at $3k + 3$), some neighbor x_k becomes faithful w.r.t j by the induction hypothesis. This x_k provides faithful w.r.t. j distance estimate value k. Thus, starting at time $(3k + 3) + 1$, any distance estimate, which is *higher* than $k + 1$, cannot be "adopted" at x_{k+1}. This is correct due to the BF minimal hop tree construction scheme used by the algorithm. See the definition of predicate $inconsis_{x_{k+1},x_k}(j)$ (Fig.1). Moreover, by Lemma 1, starting at time $k + 2$, $\texttt{dist}_{x_{k+1}}[j] \geq k + 1$. This implies the following:
(*) Starting at time $(3k + 3) + 1$, $\texttt{dist}_{x_{k+1}}[j] = k + 1$ is the only candidate distance estimate value that can be "adopted" at x_{k+1}.

If starting at time $(3k + 3) + 1$, node x_{k+1} adopts (faithful) estimates from some node z_k, such that $\mathbf{dist}(z_k, j) = k$, then the lemma holds by the induction hypothesis (and the assumption that this is a fault free interval). Let us assume, by way of contradiction, that at some time after $(3k + 3)$ the estimates at node x_{k+1} are not faithful w.r.t. j. Consider the first time $\tau > 3k + 3$ that this happens. First, note that if just before time τ, node x_{k+1} is not consistent w.r.t. j then x_{k+1} resets it variables for the tree of j. Hence, at time τ, node x_{k+1} adopts unfaithful estimates from some node y, such that $\mathbf{dist}_y[j] = dist_y > k$. On the other hand, if just before time τ node x_{k+1} does not reset its estimates, then there exists a neighbor y such that x_{k+1} is consistent with y w.r.t. j. As before, by our assumption that the estimates are unfaithful, and by the induction hypothesis, $\mathbf{dist}_y[j] = dist_y > k$.

By Lemma 1, starting at time $dist_y + 1$ ($> k + 1$), $\mathtt{dist}_y[j] > k$. If x_{k+1} either adopts estimates from y, or is consistent with y w.r.t. j, it follows that $\mathtt{dist}_{x_{k+1}}[j] = dist_y + 1 > k + 1$. However, this is impossible by statement (*) above. A contradiction. ▮

Let us now describe the ideas behind the following lemma Lem. 3 informally. Note that if a node v_0 is inconsistent w.r.t. some j, then v_0 resets its estimates for j. Unfortunately, it is possible for a node to be unfaithful w.r.t. j, while not being inconsistent. For example, consider an unfaithful w.r.t. j node v_1 that is a neighbor of v_0, such that $\mathtt{par}_{v_1}[j] = v_0$, $\mathtt{dist}_{v_1}[j] = \mathtt{dist}_{v_0}[j] + 1$, and $\mathtt{val}_{v_1}[j] = \mathtt{val}_{v_0}[j]$. (Moreover, assume that $\mathtt{dist}_{v_0}[j]$ is the smallest among the distances estimates for j received from v_1's neighbors.). Clearly, node v_1 is consistent with v_0 w.r.t. j. Hence, no resetting of the estimates for j will take place until different estimates are received.

We say that v_0 and v_1 are in an *unfaithful parent chain* (w.r.t. j). The definition of a parent chain is deferred to the full paper. It takes into account the facts that the chain can change in time, and that it can be based on `cand_par`, `cand_val`, and `cand_dist`, not just on `par`, `val`, and `dist`.

The first crucial observation is that the maximum length of an unfaithful parent chain immediately after the faults is $O(f)$. Moreover, the first node v_0 in an unfaithful parent chain is always inconsistent, hence it leaves the parent chain within $O(1)$ time, since it resets its estimates for j. Every child of v_0 (in the tree of j) now becomes a first node in an unfaithful parent chain, and hence inconsistent. Thus, it leaves the parent chain in $O(1)$ time, and so forth.

The second observation is that the number of unfaithful w.r.t. j strong messages in buffers or over links of the parent chain immediately after the faults is $O(f)$ too. Moreover, no new unfaithful w.r.t j strong messages can be created (unless additional faults occur), since j is the only node who can generate its strong messages. Moreover, for a node not in the parent chain to join a parent chain, the number of unfaithful strong messages must decrease by one (the first such message to be received by such a node is consumed, not forwarded; this is the essence of Power Supply). The end result is that some nodes may join and leave a chain several times. Nevertheless, the total number of such joins (total over of all the nodes, for a given chain) is bounded by $O(f)$. Moreover, the total number of nodes' joins, plus the original length of the chain is $O(f)$.

Finally, it is easy to observe that chains cannot merge, nor can a chain contain a cycle at any given time. The end result of these three observations is that an unfaithful parent chain disappears in $O(f)$ time. The above argument is used in the proof of the following lemma. The formal proof is deferred to the full paper.

Lemma 3. *Let $i \in V$ be any node. Let $j \in V$ be non-faulty, and assume that $\mathtt{val}_j[j]$ does not change for $t \geq 3 \cdot \min(\mathbf{depth}(i) + 1, f + 1)$ time (since a start state s_0 in $t_0 = 0$). Assume that no faults occur in the time interval $[0, t]$. Then, at time t, we have that $\mathtt{val}_i[j] \in \{\mathtt{val}_j[j], \perp\}$.*

We note that one of the main by-products of the lemma's proof is the basic property of the power supply: unfaithful estimates are forwarded only a few times (depending on f). The dependence on f, we prove, is required for the time-adaptive solution. Although [1] concentrated on the worst case stabilization time complexities (rather than time adaptivity), the proofs of [1] already hints of time adaptivity.

We can now prove that the output stabilizes quickly, provided that the non-faulty input bits remain fixed (we prove this in Theorem 5, [12]). The proof follows directly from the last two lemmas.

Theorem 3. *Starting from an arbitrary state with a fault number $f < n/2$, if non-faulty input values do not change, then starting at time* $\min(3 \cdot \mathbf{diam}, 6f) + 3$ *the output stabilizes, i.e., all output values are equal to the input values of non-faulty nodes.*

The following theorem implies the required output stabilization time in the case of $f \geq n/2$, provided that input values stabilize in $O(\mathbf{diam})$ time (Sec. 5). The proof is easily implied by Lemma 2.

Theorem 4. *Starting from an arbitrary state with a fault number $f \leq n$, if input values do not change, then starting at time* $3 \cdot \mathbf{diam} + 3$ *the output stabilizes, i.e., all output values are equal to the majority value of the input values.*

The complete state stabilization time of PS-RB is larger than $O(f)$. We have shown above that this does not harm the $O(f)$ output stabilization time. We note that the complete stabilization time of PS-RB is $O(\mathbf{diam})$ [1]. Hence, this does not harm the state stabilization time of the combined OS-IF algorithm either.

5 Input Fixing and Full Stabilization

Due to lack of space, we only give a brief outline of the IF protocol. For the details, see the extended version of this abstract ([12]). First, consider the Input fixing protocol of [29]. Recall that every node has two variables. The `output` variable must stabilize quickly, and hence it may change its value several times before stabilization (see [29]). The `input` variable, on the other hand, retains its value for a long time. In [29] it was *corrected* by the algorithm only when it was certain that this correction will not change a correct value to an incorrect one.

To ensure that a correct value will not be changed "too soon," [29] uses the assumption that only a minority of the processes are faulty. When coming to ensure the full stabilization, we need to change the input fixing protocol such that the reliance on a correct majority is removed. (The changes of the output stabilization described in Sec. 4 were due only to the asynchronous network model and assume an appropriate behavior of the IF.) The new IF protocol with the adaptation for the case of $f \geq n/2$ is given below.

Suppose that the majority of the nodes suffered faults, such that the input value in the majority was changed to some new value maj. The first idea is to view nodes with `input` = maj as correct nodes, and then use the output stabilization algorithm as is. The idea above needs some refinements, though. For example: had maj really been the value of a correct node v, then the `output` (not just the `input`) at v would have also equaled maj at the start state. We addressed this point (together with some related points) by being more careful in the definition of the Majority Consensus requirement (Def. 1) and requiring a node to be internally legal (Sec. 3) to be considered a part of the majority.

The main difficulty is raised by the need to ensure the assumption used in Sec. 4 that if the input value at a node is the majority value, then it is not changed, or, more precisely, it is assumed that this value is not changed for a sufficiently long time.

The new Input Fixing protocol ensures this property even when the majority input value belongs to faulty nodes. First, we use a self stabilizing synchronizer, which allows us to design the new IF for synchronous networks and than adopt this solution to work in asynchronous networks as desired. (Recall that Input Fixing cannot be time adaptive anyway [29], so a non-adaptive synchronizer does not harm its time complexity.) We then use a self stabilizing phase clock algorithm: this is a kind of a synchronizer that also keeps and advances a counter of the passing time.[5] Moreover, the phase clock algorithm ensures that time counter values at different nodes differ by at most the nodes' distance. Now, if a node either fixes its input, or finds an inconsistency, it resets its time counter to zero, and so do all the other nodes within diameter time. On the other hand, in order to fix an input, the time counter value must be much larger than the diameter. Hence, a long time passes (after the resetting) with no node fixing its input. Actually, this is somewhat more involved: after fixing its input, a node does not reset the counters immediately, but rather continues counting for a long time and only then resets, to give the other nodes the opportunity to reach the maximum of their counters and fix their inputs too.

References

1. Y. Afek and A. Bremler-Barr. Self-stabilizing unidirectional network algorithms by power-supply. In the 8th SODA, pp. 111-120, 1997.
2. Y. Afek and S. Dolev. Local stabilizer. In *Proceedings of the 5th Israel Symposium on Theory of Computing and Systems*, June 1997.
3. Y. Afek, S. Kutten, and M. Yung. Memory-efficient self-stabilization on general networks. In the 4th WDAG, pp. 15-28, 1990.
4. A. Arora and H. Zhang. LSRP: Local stabilization in shortest path routing. In *IEEE-IFIP DSN*, 2003.
5. A. Arora and M. G. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43:1026-1038, 1994.
6. B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing syncronization. In the 25th STOC, pp. 652-661, 1993.

[5] Indeed, we use two synchronizers; but we use only the phase clock property of the phase clock synchronizer. We do not use it as a synchronizer.

7. B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In the 32nd FOCS, pages 268-277, Oct. 1991.
8. J. Beauquier and T. Hérault. Fault Local Stabilization : the shortest path tree. In SRDS'02, pp. 62-69, 2002
9. D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, Englewood Cliffs, New Jersey, second edition, 1992.
10. C. Boulinier, F. Petit, V. Villain. When graph theory helps self-stabilization. In the 23rd PODC, pp. 150-159, 2004
11. A. Bremler-Barr, Y. Afek, and S. Schwarz. Improved BGP Convergence via Ghost Flushing. In *IEEE J. on Selected Areas in Communications*, 22:1933–1948, 2004.
12. J. Burman, T. Herman, S. Kutten, and B. Patt-Shamir. Asynchronous and Fully Self-Stabilizing Time-Adaptive Majority Consensus (extended version), http://tx.technion.ac.il/∼bjanna/.
13. J. M. Couvreur, N. Francez, and M. Gouda. Asynchronous unison. In the ICDCS'92, pp. 486-493, 1992.
14. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. In *Comm. ACM*, 17(11):643-644, November 1974.
15. S. Dolev, M. Gouda, and M. Schneider. Memory requirements for silent stabilization. In the 15th PODC, pp. 27-34, 1996.
16. S. Dolev and T. Herman. SuperStabilizing Protocols for Dynamic Distributed Systems. *Chicago Journal of Theoretical Computer Science*, 4, pp. 1-40, 1997.
17. S. Dolev and T. Herman. Parallel composition of stabilizing algorithms. In *WSS99 Proc. 1999 ICDCS Workshop on Self-Stabilizing Systems*, pp. 25-32, 1999.
18. S. Dolev. Self-Stabilization. *The MIT Press*, 2000.
19. S. Dolev, A. Israeli, and S. Moran. Resource bounds for self stabilizing message driven protocols. In the 10th PODC, pp. 281-294, 1991.
20. M. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM* 32, 2, pages 374-382, Apr. 1985.
21. C. Genolini and S. Tixeuil. A lower bound on dynamic k-stabilization in asynchronous systems. In SRDS'02, pp. 211-221, 2002.
22. T. Herman. Observations on time-adaptive self-stabilization. *Technical Report* TR 97-07.
23. T. Herman. Phase clocks for transient fault repair. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1048-1057, 2000.
24. S. Ghosh and A. Gupta. An exercise in fault-containment: self-stabilizing leader election. *Inf. Proc. Let.*, 59:281-288, 1996.
25. S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In the 15th PODC, 1996.
26. S. Ghosh, A. Gupta, and S. V. Pemmaraju. A fault-containing self-stabilizing algorithm for spanning trees. *J. Computing and Information*, 2: 322-338, 1996.
27. S. Katz and K. Perry. Self-stabilizing extensions for message-passing systems. In the 10th PODC, 1990.
28. S. Kutten and B. Patt-Shamir. Asynchronous time-adaptive self stabilization. In the 17th PODC, p. 319, 1998.
29. S. Kutten and B. Patt-Shamir. Time-Adaptive self-stabilization. In the 16th PODC, pp. 149-158, 1997.
30. S. Kutten and D. Peleg. Fault-local distributed mending. In the 14th PODC, 1995.
31. G. Parlati and M. Yung. Non-exploratory self-stabilization for constant-space symmetry-breaking. In 2nd ESA, pp. 26-28, 1994.
32. H. Zhang, A. Arora, Z. Liu. A Stability-Oriented Approach to Improving BGP Convergence. In SRDS'04, pp. 90-99, 2004.