Self-Stabilizing Byzantine Agreement in a Synchronous Network

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

by

Tomer Harpaz

Supervised by

Prof. Danny Dolev

School of Engineering and Computer Science
The Hebrew University of Jerusalem
Israel

October 2, 2011

Acknowledgments

First, I would like to express my deepest gratitude to Prof. Danny Dolev, my advisor, for his endless patience, support and guidance during my studies, and for coping with the difficulties I posed by combining research work with Talpiot. No doubt that without his positive attitude and belief I would have turned Byzantine myself by now.

Second, I would also like to thank (now Dr.) Ezra N. Hoch, who supervised my DANSS Lab project and also shared ideas and thoughts about this work. From Ezra I got the passion for distributed algorithms, and I was truly inspired by his deep understanding of the field.

Third, I would like to gratefully thank Michael Mashkautsan for his support during the past 2 years and for making my degree and training happen — if it was not for him, I doubt I would have managed to do this.

I would also like to thank my commanders from Talpiot and my friends from home and Talpiot for supporting me during the period of my research and for keeping up with me disappearing for weeks in order to work.

At last, I would like to thank my parents and brother for their endless support and love.

Abstract

This thesis presents an algorithm that provides self-stabilizing Byzantine agreement in a synchronous fully-connected network. The synchrony of the network, when the system functions correctly, is in the form of a common external "beat" which occurs in regular intervals, longer than the bound on message delivery time.

Our algorithm tolerates up to $\frac{n}{3}$ Byzantine failures and can recover from transient failures that bring all nodes to arbitrary memory states. After convergence from an arbitrary state, our primitive provides a reliable broadcast channel on top of the faulty network.

The convergence time and the running time of an agreement are both bounded by $O(f)$ where f is the actual number of Byzantine nodes during the execution. This infers that our algorithm has the "early stopping" property, meaning that it can converge and run in constant time in case there are no faulty nodes disturbing its execution.

Moreover, our algorithm does not strongly depend on the synchrony of the system, and uses it only match received messages to their appropriate phases. This allows our algorithm to be adapted to work in the the semi-synchronous model, which does not provide common "beats", only a bound on message delivery time. The translation to the semi-synchronous model is also discussed in this thesis, but not provided.

Hebrew Abstract

TODO: replace me

Table of Contents

Chapter 1

Introduction

One of the most basic building blocks in distributed algorithms is a reliable broadcast channel: it can be used to conceal underlying faults in the communication network and participating nodes by providing a robust communication primitive that is ensured to invoke the same messages for all correct nodes participating.

In turn, this primitive may be used to perform tasks such as agreeing on clock values in order to achieve clock synchronization (see [1]), which can in turn be used to self-stabilize other Byzantine fault-tolerant algorithms (see [2, 3]). These immediate advantages explain why implementing a fast and fault-tolerant broadcast channel is highly beneficial. The problem of creating a reliable broadcast primitive in an unreliable model is typically called the *Agreement* problem.[1]

This thesis addresses the *Agreement* problem in a highly faulty model which includes both permanent *Byzantine* failures of up to a third of the nodes, and *transient* failures that may affect all nodes up to a certain point in time. Our model is synchronous in the way it assumes there exists a global beat system common to all correct nodes, but our solution is such that can be adapted to work without a global beat system but with a constant bound on message delivery time between non-Byzantine nodes (sometimes called a *semi-synchronous* model).

[1] Sometimes the term *Agreement* is used to refer to a similar problem we will denote as *Consensus*, which involves all nodes having a specific input value and the primitive is supposed to output a single common value for all correct nodes. It is trivial to implement *Consensus* using *Agreement* by having each node broadcast its input value using the *Agreement* primitive, and then choose the majority of the outputs of all of the *Agreements*.

1.1 Contribution

This thesis introduces a primitive called *4-cast* which is a generalization of the *Gradecast* primitive first presented in [4]. *4-cast* basically extends the properties of *Gradecast* in order to use it in a model which does not provide "phase synchrony".[2]

We then use *4-cast* to solve the *Agreement* problem in a way that does not strongly depend on the synchrony of the system. Previous solutions such as [5] use a paradigm known as "pipelining" to convert a Byzantine-tolerant primitive such as Common Coin-Flipping (like that presented in [6] or [7]) to a self-stabilizing version that is executed in the background constantly and provides the algorithm with common input on each beat.

The primitives our solution uses are all "node invoked" in the way that they are executed using state-transition rules which depend only on messages received in past intervals and on timeout events. That is, the nodes do not share any common starting state, and the process begins by a specific node sending an invocation message.

This means that our solution uses the global beat system only to separate received messages to intervals for the state-transition rules, and therefore it should be relatively technical to convert it to an environment without a global beat system but with a constant bound on message delivery time between correct nodes.[3] We discuss the conversion to the *semi-synchronous* model (also called *bounded-delay* model) in Chapter 9.

Our solution works and converges in time linear in the number of faulty nodes, and has the "early-stopping" property, meaning that its running time is linear in the number of *actual* Byzantine nodes during execution, and not in the $\frac{n}{3}$ bound.

1.2 Thesis Outline

The thesis is organized in the following way: Chapter 2 describes other works related to our model and problem. The model and the *Agreement* problem themselves are described in Chapter 3. Chapter 4 describes the *4-cast* problem and the 4-CAST

[2] By "phase synchrony" we mean that nodes are aware of the global beat number, and not only the beat itself.

[3] There will be, of course, added redundancy in time, as nodes will need to wait between sending messages in order for their destination node to be able to separate them into different intervals. However, we expect this redundancy to be by a small factor.

primitive solving it. Chapter 5 describes the *Agreement* problem and our solution to it. Lastly, Chapter 6 discusses our conclusions and future work.

The proofs of correctness for the 4-CAST primitive and the AGREEMENT primitive appear as appendices in Chapters 7 and 8, and as stated above, Chapter 9 discusses the conversion to the semi-synchronous model.

Chapter 2

Related Work

2.1 Agreement Problems

The *Agreement* problem was first introduced in [8]. As discussed in the previous chapter, it is considered a fundamental problem in distributed algorithms, and as such, many variations of it have been widely researched for the past 30 years and algorithms have been developed for various models (see [9, 10]).

For the synchronous model with Byzantine faults, optimal probabilistic solutions have been found with expected constant running time (see [11, 7]) and optimal deterministic solutions have been found that execute in $O(f)$ rounds (see [12]).

In [4], the *Gradecast* primitive has been introduced and used to solve *Agreement* in expected constant time. As mentioned in the previous chapter, we use a generalization of *Gradecast* called *4-cast* to implement our solution. *Gradecast* was also used as a basic primitive in [13] in order to solve several variants of the *Agreement* problem in an elegant manner.

Self-stabilizing algorithms themselves, designed for models with transient faults, have also been researched thoroughly (see [14]). Designing an algorithm with this property is highly beneficial, since it has the ability to continue functioning even after suffering from a complete disruption of its normal working conditions.

2.2 Self-Stabilizing Algorithms Tolerant to Byzantine Failures

Designing algorithms that are both self-stabilizing and Byzantine fault-tolerant poses a special challenge: the usual methods used to tolerate Byzantine faults make assumptions on the initial states of the correct nodes, and the usual methods used to stabilize after transient faults requires trusting the others nodes. As a result, very few algorithms have been designed for this model, and the few that were are typically both complicated and have long convergence and running times (see [2] for a review).

An expected constant time clock synchronization algorithm for the synchronous model with both Byzantine and transient faults was described in [5]. However, it strongly depends on the synchrony of the system, and thus cannot be generalized to the semi-synchronous model.

A linear time agreement algorithm for the semi-synchronous model (with transient and Byzantine faults) was described in [15] but has a flaw. Other linear time solutions to similar problems in the semi-synchronous model (e.g. [16]) require a preexisting Agreement primitive. Therefore, further work of generalizing the algorithm presented in this thesis to the semi-synchronous model will provide a needed Agreement primitive for the semi-synchronous model that will operate in optimal deterministic time.

Chapter 3

Model and Problem Statement

This chapter contains the formal definition of the model in which the *Agreement* problem is defined and solved, and the definition of the problem itself.

3.1 Model

The definitions presented below are similar to those of [5] and those appearing in standard literature. There are, however, a few significant differences in notations and terms. Therefore, the familiar reader is advised to continue reading this chapter as well.

Our distributed model consists of a fully connected network with n nodes $\mathcal{P} = \{P_1, \ldots, P_n\}$. Communication is done using message passing, and all nodes are connected to a global beat system, which generates beats on regular intervals.

After each beat, every node processes the messages it has received up to the beat, and sends out new messages. There is a bound on the message processing and delivery time, which assures that messages sent after a specific beat will reach their destination before the next beat. We will therefore call the time interval between two consecutive beats a *round*, and subsequently refer to the flow of the algorithms using this term: in a specific round nodes process messages sent by others in the previous round.

We use the notation r to denote the external beat number, which the nodes are not aware of. We use this notation in order to define timeliness properties and ease their analysis. The round between beats r and $r+1$ is denoted as round r. Moreover, we assume that each node p keeps a local beat number ρ_p which is incremented on each beat and is used by the node to compare relative times of events.

In order to distinguish between the global round number and a node's local round number we will use r for the former and ρ for the latter. We also define the external functions $rt(\rho_p)$ that returns the global round number when the local round number of a non-faulty node p read ρ_p at the current execution; and $lt(r, p)$, which is the inverse function and represents the local round number of a non-faulty node p s.t. $rt(lt(r, p)) = r$.

Nodes in our model may be subject to two types of faults:

(1) Byzantine faults:

A percentage of the nodes may be *Byzantine*, and behave arbitrarily. We assume an information theoretic adversary with private channels and access to all communication in the network. Moreover, the non-faulty nodes cannot use any computational assumptions (e.g. signatures and/or encryption) to protect against the adversary.

We denote the bound on the number of Byzantine nodes as t, and require that $t < \frac{n}{3}$. For complexity analysis purposes, we denote the actual number of Byzantine nodes in a specific execution as $f \leq t$.

We will use the terms *Byzantine* and *faulty* interchangeably to refer to the Byzantine nodes, and use the terms *non-faulty* and *correct* to refer to the non-Byzantine nodes.

Note that we assume n and t to be fixed constants, and thus they are used as part of the algorithm and will not be changed by transient faults.

(2) Transient faults:

In addition to the Byzantine nodes, all nodes may undergo transient faults that leave their memory in an arbitrary state. Therefore, algorithms solving the problems we present will be required to converge and provide the properties required by the problem starting from any memory state and after a bounded number of rounds.

We denote by ι_0 the number of the global beat after which there are no more transient faults, and say that after beat ι_0 the system is *coherent*. Moreover, after ι_0 the set of Byzantine nodes cannot change.

Observe that the external functions $rt(\cdot)$ and $lt(\cdot,\cdot)$ are only meaningful from global round ι_0 and on. Also, observe that since we assume transient faults, the local time at a node may wrap around. However, the problems and primitives presented below require only measuring intervals of time and so it is assumed that the local time wrap-around is larger than a constant factor of the maximal interval of time needed to be measured. This way a non-faulty node can uniquely measure any necessary interval of time.

As stated above, once the system is coherent, all messages sent in round $r \geq \iota_0$ will be received before the beginning of round $r+1$. In the description of the algorithms in this thesis, we use the term "received" regarding messages to denote "received between the last beat and the one before" — that is, sent during the previous round.

Since the transient faults might destroy any kind of synchrony among the nodes besides the global beat system, the problems described in this thesis cannot require that their solutions return values to an upper layer at the end of execution (since it cannot be locally known when an execution begins or ends). Therefore, the primitives are continuously executed and notify the upper layer that an event has occurred using a local *signals* mechanism, which allows signals to be raised and caught.

For ease of reading we use different notations for signals and messages: messages are enclosed between angled brackets, e.g. $\langle \textsc{example-}msg, v_1, v_2 \rangle$, while signals are enclosed between braces, e.g. $\{\textsc{example-sig}, v_1, v_2\}$. Also, as shown in these examples, the typeface of the labels of messages and signals differ as well: both have prefixes in small-caps, but messages have names in normal text while signals have names in small-caps.

Unlike the term "received" regarding messages, when using the term "raised" regarding signals we denote "raised in the current round (previously to executing the current instruction)".

Lastly, some more notations: we denote by $\lfloor v \rfloor_r$ the value of a variable v in the beginning of global round r (during global beat r), and by $\lceil v \rceil_r$ the value of v in the end of global round r (during global beat $r+1$). Observe that $\lceil v \rceil_r = \lfloor v \rfloor_{r+1}$.

We also denote the group of rounds $\{r_1, r_1 + \Delta_{4c}, r_1 + 2\Delta_{4c}, \ldots, r_2\}$ as the round interval $[r_1, r_2]_{\Delta_{4c}}$ (assuming $r_2 = r_1 + k\Delta_{4c}$, $k \geq 0$). Δ_{4c} is a constant that will be later defined

3.2 The *Agreement* Problem

As mentioned, the problem we address in this thesis is known in standard distributed algorithms literature as the *Agreement* problem and is described as follows:

Definition 3.1 (*Agreement*)**.** A protocol for n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ where a distinguished leader $G \in \mathcal{P}$ wishes to broadcast an initial value m solves the *Agreement* problem if it conforms with the following two conditions:

> **Agreement** If a correct node signals that G broadcasted a certain value, all correct nodes will signal the same.
>
> **Validity** If the leader is correct, all correct nodes will signal that it has broadcasted m.

A more technical and rigorous definition of the problem is given in Chapter 5

Chapter 4

The 4-CAST Primitive

4.1 Chapter Outline

This chapter presents the *4-cast* problem, which is a building block we use in the solution of the *Agreement* problem in the next chapter. It also describes an algorithm which solves the problem in the given model, denoted as the 4-CAST primitive.

The problem is described in Section 4.2. Section 4.3 describes the similarity and added value of *4-cast* over the *Gradecast* problem first introduced in [4], and to which it is closely related. The algorithm of the 4-CAST primitive is provided in Section 4.5, and a more intuitive description of its flow is given in Section 4.4.

4.2 The *4-cast* Problem

An algorithm solving *4-cast* is a protocol for n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, in which a distinguished leader $G \in \mathcal{P}$ wishes to send a value v to all other nodes. An algorithm solving the *4-cast* problem is designed to tolerate or identify Byzantine initiators by appending a confidence level the signals it raises.

That is, *4-cast* is in fact a graded relaxation of a broadcast channel — in every round, each non-faulty node p may raise signals of type $\{\text{4C-ACCEPT}, G, v_p, c_p\}$ to indicate that it believes with confidence $c_p \in \{0, 1, 2\}$ that G has 4-casted the value v_p , Δ_{4c} rounds ago (Δ_{4c} depends on the solution).

In Section 4.3 we will explain why the "option" a non-faulty node has to **not** raise a signal, is in fact like adding a fourth grade of -1 to *Gradecast*.

As the problem is defined for a model with transient faults, a solution for it must also provide a constant Δ_{stb}^{4c} s.t. $\iota_1 = \iota_0 + \Delta_{stb}^{4c}$ is defined to be the round by which the system became 4C-stable. From round ι_1 and on the properties required by the problem must hold.

Finally, an algorithm solving the *4-cast* problem is one which provides an operation START4-CAST(v), raises signals of type $\{$4C-ACCEPT$, G, v, c\}$ s.t. $c \in \{0, 1, 2\}$, and satisfies the following properties:

(1) **Validity**: Had a non-faulty node G invoked START4-CAST(v) in r, $r \geq \iota_1$, then every non-faulty node p will raise $\{$4C-ACCEPT$, G, v, 2\}$ in $r + \Delta_{4c}$.

(2) **Value Agreement**: For every two non-faulty nodes p, q that have raised $\{$4C-ACCEPT$, G, v_p, c_p\}$ and $\{$4C-ACCEPT$, G, v_q, c_q\}$, respectively, in the same global round r s.t. $r \geq \iota_1$, if $c_p > 0$ and $c_q > 0$ then $v_p = v_q$.

(3) **Confidence Agreement**:

   (a) For every two non-faulty nodes p, q that have raised $\{$4C-ACCEPT$, G, v_p, c_p\}$ and $\{$4C-ACCEPT$, G, v_q, c_q\}$, respectively, in the same global round r s.t. $r \geq \iota_1$, then $|c_p - c_q| \leq 1$.

   (b) Had a non-faulty node p raised $\{$4C-ACCEPT$, G, *, c_p\}$ with $c_p > 0$ in global round r s.t. $r \geq \iota_1$, then every non-faulty node will raise $\{$4C-ACCEPT$, G, *, *\}$ in r.

(4) **Unforgeability**: Had a non-faulty node p raised $\{$4C-ACCEPT$, G, v, c_p\}$ for a non-faulty G in r s.t. $r \geq \iota_1$, then G has invoked START4-CAST(v) in $r - \Delta_{4c}$.

(5) **Uniqueness**: A non-faulty p will not raise more than one $\{$4C-ACCEPT$, G, *, *\}$ signal per $G \in \mathcal{P}$ in r, for $r \geq \iota_1$.

4.3 Motivation

The *Gradecast* problem, as described in [4], is designed for a model without transient faults. In that model, besides being synchronized, all nodes have the same "phase" (they are all aware of the global round number). Therefore, it was possible to design *Gradecast* in a way that assumed all nodes to "know" when to expect each phase of the algorithm, and to return a value-confidence pair in the same global round.

In our model, the nodes may have "stepped out of phase" because of transient faults, and so they need to try and identify when an instance is running in the network and decide whether to "join in", raise a signal, or do something else. Trying to solve *Gradecast* in our model could result in 0-graded signals being raised for non-faulty nodes which simply did not invoke a Gradecast Δ_{gc} rounds ago, but were "framed" by the Byzantine nodes. As our goal is to use *4-cast* to castrate or identify the Byzantine nodes in the *Agreement* algorithm, it is intolerable to identify non-faulties as Byzantine.

As mentioned in the previous section, in *4-cast* a non-faulty node has the option to **not** raise a signal, which is in fact a fourth possible grade. If we will denote this fourth possibility as confidence -1, we may note that property (b) of *confidence agreement* is in fact an extension of property (a): had a non-faulty node had confidence -1, no non-faulty node will have confidence > 0.

As a whole, note that *4-cast* essentially turns Byzantine errors to fail-stop errors: had a 4C-Accept signal been raised about a faulty node with confidence ≥ 1 — either all non-faulties raised it with the same value or they all have confidences in $\{0, 1\}$ and know that this node is faulty. As said, the fourth grade is required to ensure that in case a non-faulty node p did **not** invoke Start4-cast, the Byzantine will not be able to confuse the others to raise $\{$4C-Accept$, p, *, c\}$ with $c \in \{0, 1\}$. Signals about non-faulty nodes will only have confidence 2 or -1.

4.4 Intuition for the Algorithm

The algorithm is based on four 1-round phases:

(1) **Phase 1**: The Start4-cast procedure was invoked by an upper layer for some distinguished leader G with some value m. As a result, G sends a 4C-*init* message with value m to all other nodes.

(2) **Phase 2**: Nodes which have received a 4C-*init* message (at most one per G per round) respond by echoing it to all others encapsulated in a 4C-$flood$ message.

(3) **Phase 3**: Nodes which have received at least $n - t$ 4C-$flood$ messages for some (G, v_G) pair respond by echoing it to all others encapsulated in a 4C-*support* message.

(4) **Phase 4**: Nodes may raise appropriate 4C-ACCEPT signals regarding G according to rules which inspect the 4C-*support* messages received and an internal state. These rules will be described shortly.

It is important to note that since this algorithm is designed for an environment with transient faults, it is essentially rule-based: in the beginning of each round, every non-faulty node executes a set of procedures in a predefined order, in which it inspects the messages it has received during the previous round and responds by changing its internal state and/or by sending messages.

As for the rules used to invoke 4C-ACCEPT signals in phase 4:

(1) Had a node received at least $n - t$ 4C-*support* messages for some (G, v_G) pair, it will raise the signal with v_G and confidence 2.

(2) Had a node received at least $t + 1$ 4C-*support* messages for some (G, v_G) pair, it will raise the signal with v_G and confidence 1.

Had none of these rules held, we need to know whether or not to raise a signal at all (with confidence 0). Moreover, we need to make sure that the difference between confidences raised by every two non-faulty nodes will not exceed 1.

From the above rules, we have that had any non-faulty node raised a 4C-ACCEPT signal with confidence 1 for (G, v_G), at least one non-faulty node has sent a 4C-*support* message for (G, v_G) one round before. From the description of phase 3, we have that this node must have received at least $n - t$ 4C-$flood$ messages for the pair (G, v_G) and so **every** non-faulty node must have received at least $t + 1$ of these (since $n \geq 3t + 1$).

By setting a *timeout-next-round* event upon receiving at least $t + 1$ 4C-$flood$ messages in phase 3, we can assure that it will expire in phase 4 if any non-faulty will have confidence of at least 1. Hence, in phase 4, we will raise with confidence 0 if *timeout* expires, and not raise at all if it does not.

The detailed proof of correctness shows why all of these rules together assure that the properties required by the problem hold.

Lastly, in addition to the rules described above, a procedure 4C-CLEANUP is executed by every non-faulty node in the beginning of each round in order to reset possible invalid internal states created by the transient faults.

4.5 The 4-CAST Algorithm

An algorithm which solves the *4-cast* problem is detailed in Algorithm 4.1.

Our algorithm requires that a non-faulty sender G will also conform with the following criteria when invoking START4-CAST:

Definition 4.1 (4-CAST Sending Validity Criteria)**.** G does not invoke START4-CAST more than once every Δ^{4c}_{wait} rounds.

Note that although this generally prevents a node from 4-casting multiple messages at the same time, it is possible to run multiple instances of the primitive in parallel by appending unique indices representing the parallel instance index to the messages. Therefore, if the number of multiple messages required is bounded (which is the case for every realistic application) it is possible to replace every message passing with a 4-cast (e.g. by using UUIDs [17]). When using multiple parallel instances of 4-CAST in our solution to the *Agreement*, we assume that this solution is transparently applied.

As mentioned in the last section, there are $n+1$ procedures in the algorithm that are executed in every round: CLEANUP and 4C-JOININ(G) for every $G \in \mathcal{P}$. Note that CLEANUP is executed first, and then 4C-JOININ(G) is executed for every $G \in \mathcal{P}$ (possibly in parallel).

The proof of correctness of the algorithm is detailed in Chapter 7.

4.6 Complexity Analysis

As proved in Chapter 7, the 4-CAST algorithm presented in the previous section solves the *4-cast* problem with $\Delta^{4c}_{stb} = \Delta_{4c} = 3$. Hence, it finishes a valid 4-cast in 3 rounds, and converges from an arbitrary state in 3 rounds.

As for message complexity, in every round each node can send up to n messages regarding every on-going 4-cast, summing to a total maximum of $(n-1) \cdot n^2 = O(n^3)$ messages per round. As for the number of messages needed to complete a valid 4-cast initiated by a non-faulty leader, it is exactly $1+n^2+n^2 = O(n^2)$ messages accumulated over the four phases.

Algorithm 4.1 4-CAST

```
1: procedure START4-CAST(m)                                   ; executed by a sender G
2:     send ⟨4C-init, m⟩ to all                               ; phase 1
3: end procedure

4: procedure 4C-JOININ(G)                                     ; executed by p in every round for every G ∈ 𝒫
5:     if received ⟨4C-flood, G, m⟩ from ≥ t + 1 nodes then   ; phase 3
6:         if received ⟨4C-flood, G, m⟩ from ≥ n − t nodes then
7:             send ⟨4C-support, G, m⟩ to all
8:         end if
9:         timeout_p(G) ← ρ_p + 1
10:    end if
11:    if received ⟨4C-init, m⟩ from G and ρ_p > timeout_p(G) then   ; phase 2
12:        send ⟨4C-flood, G, m⟩ to all
13:        timeout_p(G) ← ρ_p + 2
14:    end if
       let maj be the value received the most amongst the 4C-support messages received; phase 4
       let #maj be the number of occurrences of maj
15:    if #maj ≥ n − t then
16:        raise {4C-ACCEPT, G, m, 2}
17:    else if #maj ≥ t + 1 then
18:        raise {4C-ACCEPT, G, m, 1}
19:    else if ρ_p = timeout_p(G) then
20:        raise {4C-ACCEPT, G, ⊥, 0}
21:    end if
22: end procedure

23: procedure 4C-CLEANUP                                      ; executed by p in the beginning of every round
24:    if timeout_p(G) > ρ_p + 1 then
25:        timeout_p(G) ← −∞
26:    end if
27: end procedure
```

Chapter 5

The AGREEMENT Primitive

5.1 Chapter Outline

This chapter gives the formal description of the *Agreement* problem described in Section 3.2. It also describes an algorithm that solves the problem in the given model, denoted as the AGREEMENT primitive.

The problem is described in Section 5.2, while Sections 5.3 and 5.4 provide an intuitive description of the algorithm flow and the formal description of the algorithm. Section 5.5 discusses the time and message complexity of the algorithm.

5.2 The *Agreement* Problem

An algorithm solving *Agreement* is a protocol for n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, in which a distinguished leader $G \in \mathcal{P}$ wishes to send a value v to all other nodes. An algorithm solving the problem provides a broadcast channel in which any distinguished node G may broadcast any value m.

As in the previous chapter — since the problem is defined for a model with transient faults, a solution must also provide a constant Δ^a_{stb} s.t. $\iota_1 = \iota_0 + \Delta^a_{stb}$ is defined to be the round in which the system became A-stable. From round ι_1 and on the properties required by the problem must hold.

An algorithm solving the *Agreement* problem is one which provides an operation STARTAGREEMENT(m), raises signals of type $\{$A-ACCEPT$, G, m, \rho_0\}$, and satisfies the following properties:

(1) **Validity**: Had a non-faulty G invoked STARTAGREEMENT(m) in r_0 s.t. $r_0 \geq \iota_1$, then every non-faulty node p will raise $\{\text{A-ACCEPT}, G, m, \rho_0^p\}$ in $r_0 + \Delta_{valid}$, s.t. $rt(\rho_0^p) = r_0$.

(2) **Agreement**: Had a non-faulty node p raised $\{\text{A-ACCEPT}, G, m, \rho_0^p\}$ in global round r s.t. $r \geq \iota_1$, then every non-faulty node q has raised $\{\text{A-ACCEPT}, G, m, \rho_0^q\}$ within the global round interval $[r - \Phi,\ r - \Phi + \Delta_{agr}]$ (for some $\Phi \in [0,\ \Delta_{agr}]$), s.t. $rt(\rho_0^q) = rt(\rho_0^p)$.

(3) **Unforgeability**: Had a non-faulty node p raised $\{\text{A-ACCEPT}, G, *, \rho_0^p\}$ in global round r s.t. $r \geq \iota_1$ and G is non-faulty, then G has invoked STARTAGREEMENT in global round $rt(\rho_0^p) = r - \Delta_{valid}$.

(4) **Termination**: Had a non-faulty node p been executing RUNNINGLOOP(G) in the end of global round r s.t. $r \geq \iota_1$, then p must have restarted RUNNINGLOOP(G) in some round r' s.t. $r - \Delta_{term} \leq r' \leq r$.

Like Δ_{stb}^a, the constants $\Delta_{valid}, \Delta_{agr}$ and Δ_{term} are properties of the algorithm.

Essentially, the definition of the problem assures that if a non-faulty node asserts that some G broadcasted m in global round r_0, all others will assert the same within a Δ_{agr} interval. Moreover, the Byzantine nodes will not be able to forge a broadcast of a correct node, and will not be able to make a correct node run RUNNINGLOOP indefinitely (RUNNINGLOOP(G) is a procedure that keeps the state of an on-going broadcast by G, and is not invoked automatically each round).

5.3 Intuition for the Algorithm

In the highest level, a node executing our algorithm tries to identify on-going broadcast agreements either by receiving an A-*init* 4-cast from the leader, or by receiving enough A-*share* 4-casts from nodes already taking part in the broadcast. Nodes that decide to "join in" on a broadcast agreement initiated by G start executing RUNNINGLOOP(G) which sends appropriate A-*share* 4-casts to others. When reaching a certain assurance level, a node will raise an A-ACCEPT signal and stop executing RUNNINGLOOP.

In fact, the AGREEMENT primitive never uses direct message-passing to pass messages, but use the underlying 4-CAST primitive for all communication. This allows nodes to be certain that if they triggered a rule of type "received X appropriate 4-casts

with confidence 2", all other correct nodes will trigger the rule "received X appropriate 4-casts with confidence ≥ 1".

Hence, for *agreement* to hold, our algorithm needs only to make sure that a node executing RunningLoop(G) will raise an A-Accept signal only when it will be certain that all other correct nodes will raise this signal as well within the Δ_{agr} interval. In fact, this works by setting the "join in" condition to receiving at least $n-t$ appropriate 4-casts with confidence ≥ 1, and setting the "raise signal" condition to receiving at least $n-t$ appropriate 4-casts with confidence 2.

As stated in Section 4.4, it is important to note that since this algorithm is designed for an environment with transient faults, it is essentially rule-based: in the beginning of each round every non-faulty node executes a set of procedures in a predefined order, in which it inspects the 4-casts it has received this round and responds by changing its internal state and/or by sending more 4-casts. To provide the communication infrastructure, the procedures of the 4-cast primitive are executed in the beginning of each round in the background. Technically, the Agreement primitive inspects the 4-casts received in the current round by inspecting the 4C-Accept signals raised by the underlying 4-cast primitive.

The main difficulty in solving the *Agreement* problem is conforming to both the *agreement* property **and** the *validity* property. The main idea presented a few paragraphs above is almost enough to provide *agreement* — a correct node p will only agree once it will be sure from the confidence levels of the 4-cast messages it received and from the dynamics of the algorithm that all other nodes will subsequently agree as well. However, in case G is Byzantine, it might wait until p agrees and then send a new A-*init* message to the other nodes and "pull them out" of the the previous agreement right before they raised their signals (but after p did). Hence, to provide *agreement* we need to add a second rule that will make nodes participating in an agreement by G ignore new A-*init* messages from G in the mean while.

This new rule, however, is now possibly conflicting with *validity* — what if in a different scenario the Byzantine nodes trick some correct nodes to start executing RunningLoop(G) for a correct G, just before G actually tried to invoke a broadcast agreement? The correct nodes already executing will not take part, possibly breaking *validity*!

The solution we found is to use 4-cast to essentially turn the Byzantine errors to fail-stop errors, as explained in Section 4.3. The definition of *4-cast* assures that 4-casts sent by Byzantine node are either received with the same value for all correct

nodes, in which case they cannot cause any disruption, **or** the faulty sender will be identified and "burned" because its 4-cast will be received with confidence 0 or 1 by all correct nodes. Correct nodes can then subsequently ignore nodes identified as Byzantine and intuitively after t disruptions all faulty nodes will be ignored by all correct nodes.

As presented above, our problem was that Byzantine nodes could trick correct nodes to execute RUNNINGLOOP(G) for a correct G without G explicitly invoking an agreement. We will later prove that in our algorithm, the Byzantine adversary will have to "burn" (identify and make ignored by the correct nodes) at least one faulty node per 2 rounds it makes a non-faulty node execute RUNNINGLOOP(G) without G's consent. Therefore, making sure that a correct G will not invoke STARTAGREEMENT more than once every $O(2f)$ rounds will make sure no correct nodes will be running RUNNINGLOOP(G) at that time.

We will now describe the flow of the algorithm in more depth, and point out the important details and how they correspond to the ideas presented above.

As we have already explained, our algorithm uses 4-CAST as its communication infrastructure and ignores nodes it identifies as Byzantine. In practice this is done by continuously executing two copies of the 4-CAST primitive in parallel in the background: one is the normal 4-CAST presented in the previous chapter, and the other is a wrapper we denote as $\overline{\text{4-CAST}}$ which ignores the nodes that were in the set BAD in the beginning of each round. The exact details of $\overline{\text{4-CAST}}$ are described in Algorithm 5.1, and as can be seen there, we denote the signals raised by $\overline{\text{4-CAST}}$ as $\overline{\text{4C-ACCEPT}}$, and those of the original 4-CAST remain 4C-ACCEPT.

In fact, the AGREEMENT primitive uses the 4C-ACCEPT signals only to identify Byzantine nodes and add them to the BAD list, and uses the $\overline{\text{4C-ACCEPT}}$ signals for all of the state-transition rules. As we will prove in Corollary 8.3, the properties required by *4-cast* hold for the $\overline{\text{4C-ACCEPT}}$ signals as well after a certain point in time. In addition, it is important to remove nodes from the BAD list after a period of time, or else correct nodes may be kept there from the transient faults phase.

The actual bookkeeping of the BAD list is done in the CLEANUP and SNITCH procedures, and can be seen in detail in the next section.

The next subtle detail is that we replaced message passing with 4-casts, but 4-casts take Δ_{4c} rounds to complete rather than 1. Therefore, in order to make sure nodes executing RUNNINGLOOP(G) at the same time will be able to communicate coherently, we make sure that they are kept "in phase": meaning that they all initiate 4-casts in

Algorithm 5.1 $\overline{\text{4-CAST}}$

```
1: procedure 4C-JoinIn(G)
2:     for every q ∈ BAD_p do
3:         ignore messages received from q when executing the following
               instruction (Line 5)
4:     end for
5:     execute 4C-JoinIn(G) and catch raised signals to the set S
6:     for every {4C-Accept, G, v, c} signal in S do
7:         if G ∉ BAD_p then
8:             raise {4C-Accept, G, v, c}
9:         else
10:            pass
11:        end if
12:    end for
13: end procedure
```

Figure 5.1: The $\overline{\text{4-CAST}}$ wrapper for 4-CAST, as executed by a node p

the same global rounds and expect 4-casts to finish in the same global rounds. In fact, we define these "valid" global rounds to be $r_0 + k\Delta_{4c}$, $k \in \mathbb{N}$ where r_0 is the round in which G supposedly invoked StartAgreement and sent the first 4-cast regarding the on-going agreement.

Therefore, a node executing RunningLoop(G) — that is, participating in an agreement for a value broadcasted by G — will always be participating with respect to some alleged invocation time r_0. The value of $lt(r_0, p)$ will be kept in the $started_p(G)$ variable, and will only be set when "restarting" RunningLoop(G). Every A-*share* message passed between correct nodes will also contain the pair of G and the "age" of the broadcast by the time the message was sent — how many rounds passed since r_0.

This way, we can make sure that nodes will restart RunningLoop only "in phase" with nodes already executing it, and that correct nodes will know to which specific broadcast invocation an exchanged message is referring to. We will subsequently use the term "in phase" regarding nodes that are executing RunningLoop(G) with $started(G)$ values that correspond to the same global round (this term is accurately defined in Definition 8.1).

We will now describe what is done in the JoinIn and RunningLoop procedures, which are the heart of the algorithm:

(1) RunningLoop:

A node p executing RunningLoop(G) has three important state variables:

$started_p(G)$, $state_p(G)$ and $v_p(G)$.

$started_p(G)$ was discussed in the previous paragraphs; $v_p(G)$ is the current agreement value p holds for the on-going broadcast; and $state_p(G) \in \{0, 1, 2\}$ will be explained shortly.

RUNNINGLOOP(G) works in phases of Δ_{4c} rounds: it updates its state variable using UPDATESTATE(G) which inspects the A-*share* 4-casts received from other in-phase nodes; 4-casts a new A-*share* message to all others with its current $v_p(G)$ value; and then waits Δ_{4c} rounds to receive the next round of communication.

UPDATESTATE(G) sets $v_p(G)$ to the majority value among the in-phase A-*share* messages received with confidence ≥ 1, and increments $state_p(G)$ in case the majority value was received with confidence 2 from at least $n - t$ nodes.

In case a node has set $state_p(G) = 2$, it will raise an A-ACCEPT signal with $v_p(G)$ and $started_p(G)$ and stop RUNNINGLOOP(G).

RUNNINGLOOP(G) also uses a $timeout_p(G)$ variable to make sure it does not execute for more than one round unless it is actively "kept alive" using a specific condition in JOININ. We will later prove that the dynamics of the algorithm ensure that RUNNINGLOOP(G) will not execute indefinitely, but at most $O(f)$ rounds.

(2) JOININ:

There are two rules in JOININ, corresponding to the ones presented in the main idea in the beginning of this section:

(a) If p received an A-*init* message from G with confidence 2 **and** is not currently executing RUNNINGLOOP(G), it will restart RUNNINGLOOP(G) and set $started_p(G) = \rho - \Delta_{4c}$.

(b) If p received at least $n - t$ signals in phase with (G, r_0), all with the same value and confidence ≥ 1, then:

i. If it is not currently executing RUNNINGLOOP(G) or is executing it with a different phase, it will restart it to be in phase with r_0;

ii. If it is currently executing RUNNINGLOOP(G) in phase with r_0, it will reset its $timeout_p(G)$ variable in order to keep it alive for another Δ_{4c}-round phase.

The resulting dynamics are that if a node sets $state_p(G) = 1$, all correct nodes will be executing RUNNINGLOOP(G) in the next round (Lemma 8.12); and if a node sets $state_p(G) = 2$ all correct nodes will set $v(G)$ to the same majority value, increment $state_p(G)$, and so raise A-ACCEPT in the current round or the next (Claim 8.6).

One final detail is that after a node raises an A-ACCEPT signal, it has to ignore A-*share* messages in-phase with this broadcast for the next Δ_{4c} rounds, in order not to rejoin the broadcast using the first condition in JOININ.

5.4 The AGREEMENT Algorithm

We will now present the formal description of our solution of the *Agreement* problem. The algorithm is detailed in Algorithms 5.2 and 5.3.

Our algorithm requires that a non-faulty sender G will also conform with the following criteria when invoking STARTAGREEMENT:

Definition 5.1 (AGREEMENT Sending Validity Criteria)**.** G does not invoke STARTAGREEMENT more than once every Δ^a_{wait} rounds.

As with the 4-CAST algorithm and as mentioned in the previous section, there are several procedures in this algorithm that are executed in every round. The order of execution among these procedures in each round for every non-faulty node is:

(1) CLEANUP;

(2) The procedures of 4-CAST and $\overline{\text{4-CAST}}$;

(3) SNITCH;

(4) JOININ(G) for every $G \in \mathcal{P}$ (may be executed in parallel);

(5) Instances of RUNNINGLOOP that may be running

The proof of correctness of the algorithm is detailed in Chapter 8.

5.5 Complexity Analysis

As proved in Chapter 8, the AGREEMENT algorithm presented in the previous section solves the *Agreement* problem with $\Delta_{valid} = 2\Delta_{4c} = 6$, $\Delta_{agr} = \Delta_{4c} = 3$, and $\Delta^a_{wait} =$

$\Delta^a_{stb} = O(f)$ rounds. Hence, valid broadcasts are finished in 6 rounds, and the algorithm converges from an arbitrary state in $O(f)$ rounds (in fact, $\Delta^a_{stb} = 36f + 55$ and $\Delta^a_{wait} = 12f + 19$).

As for message complexity, every Δ_{4c} rounds each node invokes one 4-cast for every on-going AGREEMENT (in RUNNINGLOOP), summing to a total maximum of $O(n^2)$ 4-casts per round. As for the number of messages needed to complete a valid 4-cast initiated by a non-faulty leader, it takes a total of $1 + 2n$ 4-casts, which sum to $O(n^3)$ messages.

Algorithm 5.2 AGREEMENT wrapper

procedure STARTAGREEMENT(m) ; *executed by the leader G*
  START4-CAST($\langle$A-*init*$, m\rangle$)
end procedure

procedure JOININ(G) ; *executed by p in the beginning of every round for every $G \in \mathcal{P}$*
  if raised $\{\overline{\text{4C-ACCEPT}}, G, \langle$A-*init*$, m\rangle, 2\}$ **and not** running RUNNINGLOOP(G) **then** ; *init*
    $v_p(G) \leftarrow m$
    $started_p(G) \leftarrow \rho_p - \Delta_{4c}$
    start RUNNINGLOOP(G)
  end if
  if raised $\geq n - t$ signals $s_i = \{\overline{\text{4C-ACCEPT}}, p_i, \langle$A-*share*$, G, m, age\rangle, c_i\}$ with $c_i \geq 1$ for some $age = k\Delta_{4c}$, $k \geq 1$ **then**
    if running RUNNINGLOOP(G) **and** $started_p(G) = \rho_p - (age + \Delta_{4c})$ **then** ; *stay*
      $timeout_p(G) \leftarrow \rho_p$
    else ; *join*
      $started_p(G) \leftarrow \rho_p - (age + \Delta_{4c})$
      stop RUNNINGLOOP(G) if it is running; start RUNNINGLOOP(G)
    end if
  end if
end procedure

procedure SNITCH ; *executed by p in the beginning of every round*
  for every $q \in \mathcal{P}$ **do**
    if raised $\{$4C-ACCEPT$, q, \langle * \rangle, c\}$ with $0 \leq c \leq 1$ **then** ; *detect faulty nodes*
      $BAD_p \leftarrow BAD_p \cup \{q\}$
      ρ-$BAD_p(q) \leftarrow \rho_p$
    end if
  end for
end procedure

procedure CLEANUP ; *executed by p in the beginning of every round*
  for every $q \in \mathcal{P}$ **do**
    if $q \in BAD_p$ **and** ρ-$BAD_p(q) \notin (\rho_p - \Delta_{rmv}, \rho_p]$ **then** ; *forget transient faults*
      $BAD_p \leftarrow BAD_p - \{q\}$
    end if
    if $joined_p(q) \geq \rho_p$ **or** $timeout_p(q) \geq \rho_p$ **or** $started_p(q) \geq \rho_p - \Delta_{4c}$; *clean local*
        or $state_p(q) \notin \{0, 1\}$ **then** ; *transient faults*
      $joined_p(q) \leftarrow -\infty$; $started_p(q) \leftarrow -\infty$
      $timeout_p(q) \leftarrow -\infty$; $state_p(q) \leftarrow 0$
      stop RUNNINGLOOP(q)
    end if
  end for
end procedure

Algorithm 5.3 AGREEMENT running loop

39: **procedure** RUNNINGLOOP(G) ; *executed by p trying to agree in a value broadcasted by G*
40: $joined_p \leftarrow \rho_p$; *in this procedure ψ_p refers to $\psi_p(G)$*
41: $timeout_p \leftarrow \rho_p$
42: $state_p \leftarrow 0$
43: **if** $started_p > \rho_p - \Delta_{4c}$ **then**
44: UPDATESTATE(G)
45: **end if**
46: **while** $\rho_p \leq timeout_p$ **and** $state_p < 2$ **do**
47: START4-CAST($\langle$A-$share, G, v_p, \rho_p - started_p\rangle$) and wait for Δ_{4c} rounds
48: UPDATESTATE(G)
49: **end while**
50: **if** $state_p = 2$ **then**
51: START4-CAST($\langle$A-$share, G, v_p, \rho_p - started_p\rangle$)
52: ignore $\{\overline{\text{4C-ACCEPT}}, *, \langle$A-$share, G, *, \rho_p - started_p + k\Delta_{4c}\rangle, 2\}$ signals in rounds $\rho_p + (k+1)\Delta_{4c}$ for $k \in \{0, 1\}$
53: ignore $\{\overline{\text{4C-ACCEPT}}, G, \langle$A-$init, *\rangle, 2\}$ signals in the next Δ_{4c} rounds
54: raise $\{$A-ACCEPT$, G, v_p, started_p\}$
55: $state_p \leftarrow 0$; *this is for ease of proofs*
56: **end if**
57: **end procedure**

58: **procedure** UPDATESTATE(G) ; *in this procedure ψ_p refers to $\psi_p(G)$*
59: **if not** $\big(\rho_p = joined_p + k_1\Delta_{4c} = started_p + k_2\Delta_{4c}$ s.t. $k_2 > k_1 \geq 0\big)$ **then** ; *terminate*
60: stop RUNNINGLOOP(G) ; *transient runs*
61: **return**
62: **end if**

let $\langle q, v, c\rangle$ represent that we raised $\{\overline{\text{4C-ACCEPT}}, q, \langle$A-$share, G, v, \rho_p - started_p(G) - \Delta_{4c}\rangle, c\}$

let maj be the value which appears the most among those with confidence ≥ 1
let $\#maj$ be the number of occurrences of maj with confidence 2

63: **if** $v_p \neq maj$ **then** ; *update v_p*
64: $v_p \leftarrow maj$
65: $state_p \leftarrow 0$
66: **end if**
67: **if** $\#maj \geq n - t$ **then** ; *update $state_p$*
68: $state_p \leftarrow state_p + 1$
69: **else**
70: $state_p \leftarrow 0$
71: **end if**
72: **end procedure**

Chapter 6

Conclusions and Future Work

6.1 Turning Byzantine Failures to Fail-Stop

The main idea we used in the AGREEMENT algorithm is to replace all message-passing with 4-casts and use the implemented 4-CAST primitive to provide the communication infrastructure.

This incapacitated the Byzantine nodes by depriving them from the ability to confuse the correct nodes by sending them different values. The 4-cast channel ensured that in case a faulty node tried to confuse correct nodes, it would have been identified and further ignored.

We believe that this method and primitive can be used as a building block for many more algorithms, as [13] did with *Gradecast*.

6.2 Node-invoked Self-Stabilizing Byzantine Algorithms

The main contribution of the algorithm presented in this thesis is the fact that it can be translated to work in the semi-synchronous model. The general idea for the adaptation is described in Chapter 9, and is supposed to generally work for any "node invoked" self-stabilizing Byzantine-tolerant algorithm.

The concept of "node invoked" algorithms was briefly described in Section 1.1, but generally means that the algorithm is independent of any initial synchrony. If a round-based distributed algorithm communicating using message passing is "invoked"

by a single node we call it "node invoked". For example, *Gradecast*, *4-cast* and *Agreement* are node invoked, while *Consensus* is not.

6.3 Future Work

The main future work we expect is to complete the translation of the 4-CAST primitive to the semi-synchronous model and then use it to either create a general translation mechanism for node-invoked algorithms, or to directly translate our AGREEMENT primitive.

Another possible direction after translating the 4-CAST primitive, is to try and implement *VSS* or *m-VSS* and continue in the path of [11] or [7] in order to achieve *Agreement* in expected constant time in the semi-synchronous model.

Chapter 7

Appendix: Proofs for the 4-CAST Primitive

Claim 7.1. Let p be a non-faulty node and let $r \geq \iota_0$ be a global round, then $\lceil timeout_p(G) \rceil_r \leq lt(r,p) + 2$.

Proof. First, note that after executing CLEANUP, $timeout_p(G) \leq lt(r,p) + 1$.

Had p also changed $timeout_p(G)$ using 4C-JOININ, it must have been in Lines 13 or 9 and then $timeout_p(G) \leq lt(r,p) + 2$ as well. □

Corollary 7.1. Let p be a non-faulty node and let $r \geq \iota_0 + 1$ be a global round. The condition on Line 24 will not hold for p in r.

Proof. Note that by applying Claim 7.1 to $r - 1 \geq \iota_0$, we have that $\lfloor timeout_p(G) \rfloor_r = \lceil timeout_p(G) \rceil_{r-1} \leq lt(r-1,p) + 2 \leq lt(r,p) + 1$ and the condition on Line 24 will be false in r. □

Lemma 7.1. Let G be a non-faulty node and let $r \geq \iota_0 + 2$. Had a non-faulty node p set $timeout_p(G) \geq lt(r,p)$ in round r, then G has invoked START4-CAST in $rt(\lceil timeout_p(G) \rceil_r) - 3$.

Proof. Since $r \geq \iota_0$, p must have set $timeout_p(G)$ using Lines 13 or 9.

Had p set $timeout_p(G)$ using Line 13, the required follows immediately from Line 11 and the fact that $r - 1 \geq \iota_0$.

Had p set $timeout_p(G)$ using Line 9, then Line 5 implies that at least 1 non-faulty node q has sent $\langle 4C\text{-}flood, G, * \rangle$ in $r - 1$. Hence, q must have executed Line 13 in $r - 1$ and the required follows from the previous paragraph. □

Corollary 7.2. Let G be a non-faulty node and let $r \geq \iota_0 + 4$. Has a non-faulty node p had $\lfloor timeout_p(G) \rfloor_r \geq lt(r, p)$ in round r, then G has invoked START4-CAST in $rt(\lfloor timeout_p(G) \rfloor_r) - 3$.

Proof. Assume to the contrary that p has not set $timeout_p(G)$ in round $r-1$ or $r-2$, it would have $\lceil timeout_p(G) \rceil_{r-3} = \lfloor timeout_p(G) \rfloor_r \geq lt(r, p) = lt(r-3, p) + 3$ in contradiction to Claim 7.1.

Therefore, by applying Lemma 7.1 to $r-1$ or $r-2$ (both are $\geq \iota_0 + 2$) we have the required. □

Lemma 7.2. Had a non-faulty G invoked START4-CAST(v) in global round r, $r \geq \iota_0 + 3$, then the condition in Line 11 will not hold for any non-faulty node p in round $r+1$.

Proof. First, note that since $r \geq \iota_0$ G will send a $\langle 4C\text{-}init, v \rangle$ message to all other nodes in r, which will be received by all non-faulty nodes in round $r+1$.

Assume to the contrary that the condition in Line 11 will not hold for some non-faulty node p in $r+1$.

That is, $\lfloor timeout_p(G) \rfloor_{r+1} \geq lt(r+1, p)$. By applying Corollary 7.2 to p on $r+1$, we have that G has invoked START4-CAST in $rt(\lfloor timeout_p(G) \rfloor) - 3$.

From Claim 7.1 we have that $\lfloor timeout_p(G) \rfloor_{r+1} \leq lt(r+1, p) + 1$, therefore G has invoked START4-CAST in $r-1$ or $r-2$ (both $\geq \iota_0$) in contradiction with the fact that G is conforming with Definition 4.1 in r. □

Claim 7.2 (Validity)**.** Had a non-faulty node G invoked START4-CAST(v) in r_0, $r_0 \geq \iota_0 + 3$, then every non-faulty node p has raised $\{4C\text{-ACCEPT}, G, v, 2\}$ in $r_0 + 3$.

Proof. From Lemma 7.2 we have that in round $r_0 + 1$ the condition in Line 11 will hold for every non-faulty node. It follows that every non-faulty node will send $\langle 4C\text{-}flood, G, m \rangle$ to all others in $r+1$.

Hence, in round $r+2$ every non-faulty node will receive at least $n-t$ $\langle 4C\text{-}flood, G, m \rangle$ messages and pass the conditions on Lines 5 and 6. All of them will then send $\langle 4C\text{-}support, G, m \rangle$ to all others. Note that they will not enter the condition on Line 11, because p will not invoke START4-CAST$(*)$ in round $r+1$ (by conforming to Definition 4.1).

On round $r+3$ every non-faulty node will receive the 4C-*support* messages sent by all others in $r+2$, and so they will all pass the condition on Line 15 and raise $\{4C\text{-ACCEPT}, G, m, 2\}$ as required. □

Lemma 7.3. If two non-faulty nodes p, q send messages $\langle$4C-*support*$, G, m_p\rangle$ and $\langle$4C-*support*$, G, m_q\rangle$, respectively, in global round r, $r \geq \iota_0 + 1$, then $m_p = m_q$.

Proof. Assume to the contrary that $m_p \neq m_q$. Since p and q sent the said messages, they have both entered the condition on Line 6 in global round $r \geq \iota_0 + 1$. Each of them received the value it has sent as a 4C-*flood* message from a majority of $n - t$ nodes sent in $r - 1$.

Suppose $0 \leq k \leq t$ of p's messages are from faulty nodes, then q must have received m_p flooded from at least $n - t - k$ non-faulty nodes (as $r - 1 \geq \iota_0$). Also, suppose the rest of the non-faulty nodes sent $m_q \neq m_p$ to q, and all of the faulty nodes sent m_q to q. We have that q received m_q from at most $t + (n - t - (n - t - k)) = t + k \leq 2t < n - t$ nodes, where the last inequality follows from the fact that $n \geq 3t + 1$. This is in contradiction to the fact that q passed the condition on Line 6 with m_q. □

Claim 7.3 (Value Agreement)**.** For every two non-faulty nodes p, q that have raised $\{$4C-Accept$, G, v_p, c_p\}$ and $\{$4C-Accept$, G, v_q, c_q\}$, respectively, in the same global round r, $r \geq \iota_0 + 2$, if $c_p > 0$ and $c_q > 0$ then $v_p = v_q$.

Proof. According to the conditions on Lines 15 and 17, p and q had to receive the same value supported by at least $t + 1$ nodes in round $r - 1 \geq \iota_0 + 1$.

Assume to the contrary that $v_p \neq v_q$, then p has received v_p from at least one non-faulty node, and so did q for v_q. This stands in contradiction to Lemma 7.3 applied to round $r - 1$. □

Lemma 7.4. Had a non-faulty node p raised $\{$4C-Accept$, G, v_p, c_p\}$ with $c_p \geq 1$ in global round r, $r \geq \iota_0 + 2$, then every non-faulty q will have $rt(\lceil timeout_q(G)\rceil_r) = r$.

Proof. Since $c_p > 0$, p has received v_p from at least $t + 1$ nodes in r. Since $r \geq \iota_0 + 2$, at least one non-faulty node $\bar{p}$ sent $\langle$4C-*support*$, G, v_p\rangle$ in round $r - 1 \geq \iota_0 + 1$. It follows from Line 6 that $\bar{p}$ had received v_p flooded from $n - t$ nodes in round $r - 1$ and at least $t + 1$ of these floods had to be non-faulty (since $n \geq 3t + 1$ and $r - 2 \geq \iota_0$).

Therefore, in round $r - 1$, every non-faulty node had to receive the same flooded value from at least $t+1$ nodes, pass the condition on Line 5 and set $\lceil timeout_p(G)\rceil_{r-1} = lt(r, p)$. Also, no non-faulty node will execute Line 13 or send $\langle$4C-*flood*$, G, *\rangle$ in $r - 1$ because none will pass the condition in Line 11 after updating $timeout_p(G)$.

Let q be a non-faulty node. In round r q will not change $timeout_q(G)$ again, because the condition in Line 11 will be false and so will the one in Line 5 since

no non-faulty node has sent $\langle \text{4C-}flood, G, * \rangle$ in $r-1$ (and Corollary 7.1). Thus, $\lceil timeout_q(G) \rceil_r = \lceil timeout_q(G) \rceil_{r-1} = lt(r,q)$ as required. □

Claim 7.4 (Confidence Agreement 1)**.** For every two non-faulty nodes p, q that raised $\{\text{4C-ACCEPT}, G, v_p, c_p\}$ and $\{\text{4C-ACCEPT}, G, v_q, c_q\}$, respectively, in the same global round r, $r \geq \iota_0 + 2$, we have $|c_p - c_q| \leq 1$.

Proof. First, assume that either p or q has $c = 2$, and assume w.l.o.g that it is p. From the condition in Line 15, p has received v_p from at least $n - t$ nodes. Assume to the contrary that q has $c_q < 1$ (iff q has received its majority supported value from less than $t + 1$ nodes). Since $r \geq \iota_0 + 1$, at least $n - 2t$ of p's received values were from non-faulty nodes (and received by q as well). However, $n - 2t \geq t + 1$ in contradiction (since $n \geq 3t + 1$).

Hence, had a non-faulty node decided with $c = 2$, all other non-faulty nodes will decide with $c \geq 1$.

Second, assume that either p or q has $c = -1$, and assume w.l.o.g that it is p. Assume to the contrary that q has $c_q > 0$. From Lemma 7.4, p will have $\lceil timeout_p(G) \rceil_r = lt(r,p)$ and so p will satisfy the condition in Line 19 in r in contradiction to the fact that $c_p < 0$.

Thus, had a non-faulty node decided with $c = -1$, all other non-faulty nodes will decide with $c \leq 0$, concluding the proof. □

Corollary 7.3 (Confidence Agreement 2)**.** Had a non-faulty node p raised a signal $\{\text{4C-ACCEPT}, G, *, c_p\}$ in r, $r \geq \iota_1$ with $c_p > 0$, then every non-faulty node has raised $\{\text{4C-ACCEPT}, G, *, *\}$ in r.

Proof. From Lemma 7.4, every non-faulty node will have $\lceil timeout(G) \rceil_r = lt(r)$. Therefore they will all raise $\{\text{4C-ACCEPT}, G, *, *\}$ signals in r. □

Claim 7.5 (Unforgeability)**.** Had a non-faulty node p raised $\{\text{4C-ACCEPT}, G, v_p, c_p\}$ in r, $r \geq \iota_0 + 3$ for a non-faulty G, and had G not invoked START4-CAST in $r - 3$, then $c_p = -1$.

Proof. Assume to the contrary that $c_p \geq 1$. Since $r - 1 \geq \iota_0$, it follows from Lines 15 and 17 that p has received a support for v_p from at least one non-faulty node p', sent in round $r - 1$. From Line 6 and since $r - 2 \geq \iota_0$, p' had received flood messages for v_q from at least $t + 1$ non-faulty nodes, sent in round $r - 2$ (let $\bar{p}$ be one of these nodes).

From Line 11, $\bar{p}$ has received an A-*init* message from G sent in round $r-3 \geq \iota_0$, in contradiction with the assumption.

Now, assume to the contrary that $c_p = 0$. From Line 19, $\lceil timeout_p(G) \rceil_r = lt(r,p)$. By applying Corollary 7.2 we have invoked START4-CAST in $r-3$ in contradiction. □

Claim 7.6 (Uniqueness)**.** Let p be a non-faulty node, and let $r \geq \iota_0$. p will not raise more than one $\{\text{4C-ACCEPT}, G, *, *\}$ signal per $G \in \mathcal{P}$ in r.

Proof. Follows directly from the algorithm. □

Theorem 7.1 (4-CAST)**.** 4-CAST solves the *4-cast* problem for $\Delta^{4c}_{stb} = \Delta_{4c} = \Delta^{4c}_{wait} = 3$.

Proof. Follows directly from all of the previous claims. □

Chapter 8

Appendix: Proofs for the AGREEMENT Primitive

We first define some notations and add clarifications regarding phrases we will use.

We denote by $\cup BAD = \bigcup_{p \in \mathcal{P}'} BAD_p$ the union of the BAD sets of all non-faulty nodes. Similarly, we denote by $\cap BAD = \bigcap_{p \in \mathcal{P}'} BAD_p$ the intersection of the BAD variables for all non-faulty nodes.

When we say that a node p has been running RUNNINGLOOP(G) **in the beginning of** round r, we imply that RUNNINGLOOP(G) was running *after* p has executed CLEANUP in round r. When we say that a node p is running RUNNINGLOOP(G) **in the end of** r, we imply that it was executed in r and has not terminated. When we say that p has been running RUNNINGLOOP(G) **in** r, we imply that it was executed (not necessarily in the beginning, and could also have terminated).

We say that p has received a 4-cast from q with message m and confidence c in round r, iff p has raised $\{\overline{\text{4C-ACCEPT}}, q, m, c\}$ in r. Note that we only use the 4C-ACCEPT signals to identify faulty nodes, and use the $\overline{\text{4C-ACCEPT}}$ signals as "received communication". Corollary 8.3 states that this usage is valid some time after ι_0.

General claims

Definitions

Definition 8.1. We say that a non-faulty node p is *in phase with* (G, r_0) *in* r iff $rt(started_p(G)) = r_0$ and $r = r_0 + k\Delta_{4c}$, $k \geq 1$.

For example, if we say that p has been running RunningLoop(G) in phase with (G, r_0) in the beginning of r, we imply that p has been running RunningLoop(G) in the beginning of $r = r_0 + k\Delta_{4c}$, $k \geq 1$ and $rt(\lfloor started_p(G) \rfloor_r) = r_0$.

Definition 8.2. Let $r \geq \iota_0$ be a global round, and let p be a non-faulty node.

We say that an $\langle$A-*share*$, G, *, age\rangle$ message 4-casted by p in r is *in phase with* (G, r_0) if $r - age = r_0$ and $age = k\Delta_{4c}$, $k \geq 1$.

We also say that an $\langle$A-*share*$, G, *, age\rangle$ 4-cast received by p in r is *in phase with* (G, r_0) if $r - (age + \Delta_{4c}) = r_0$ and $age = k\Delta_{4c}$, $k \geq 1$.

Definition 8.3. Had a non-faulty node p returned from RunningLoop(G) in global round r, $r \geq \iota_0$, we say that it has *returned successfully* if p entered the conditional on Line 50 before returning.

Corollary 8.1. Note that using these definitions, we can "translate" some of the algorithm to a more intuitive language. Here are some of the intuitions (having r as the current global round):

(1) Line 8 starts RunningLoop(G) in phase with $(G, r - \Delta_{4c})$.

(2) Line 10 requires that we receive $n - t$ 4-casts of A-*share* messages all with the same phase (G, r_0) s.t. $r_0 \leq r - 2\Delta_{4c}$.

   (a) Line 11 requires p to be running RunningLoop(G) in phase with (G, r_0) in the beginning of r.

   (b) Line 15 resets RunningLoop(G) so that it would be in phase with (G, r_0).

(3) Line 59 makes sure that p is in phase.

We will use some of these intuitions in the proofs below.

Definition 8.4. Let p be a non-faulty node, and let $r \geq \iota_0$ be a global round. Had p received at least $n - t$ $\{\overline{\text{4C-Accept}}, *, \langle$A-*share*$, G, m, age\rangle, c_i\}$ signals s.t. $c_i \geq 1$ in round r, we say that the condition in Line 10 was satisfied for the (G, r_0, m) *triplet* in r, where $r_0 = r - (age + \Delta_{4c})$

Cleanliness

Claim 8.1. Let p be a non-faulty node, let $r \geq \iota_0$ and let $G \in \mathcal{P}$, then:

(1) $\lceil joined_p(G) \rceil_r < lt(r+1, p)$;

(2) $\lceil timeout_p(G) \rceil_r < lt(r+1, p)$;

(3) $\lceil started_p(G) \rceil_r < lt(r+1, p) - \Delta_{4c}$;

(4) and $\lceil state_p(G) \rceil_r \in \{0, 1\}$.

Proof. First, note that after CLEANUP was executed on r (and before anything else was) it granted that:

(1) $joined_p(G) < lt(r, p)$;

(2) $timeout_p(G) < lt(r, p)$;

(3) $started_p(G) < lt(r, p) - \Delta_{4c}$;

(4) and $state_p(G) \in \{0, 1\}$.

Thus, had none of these variables changed during r, the required follows immediately since $lt(r, p) \leq lt(r+1, p)$.

Had $joined_p(G)$ changed during r, then as $r \geq \iota_0$ it must have been set using Line 40 to be $lt(r, p) < lt(r+1, p)$ as required. The same goes for $timeout_p(G)$ with Lines 12 and 41.

Had $started_p(G)$ changed during r, then as $r \geq \iota_0$ it must have been set using Lines 7 or 14 to be $\leq lt(r, p) - \Delta_{4c} < lt(r+1, p) - \Delta_{4c}$.

Had $state_p(G)$ been changed during r then a $r \geq \iota_0$, it must have been using Lines 42, 55, 65 or 70 to be 0, or incremented using Line 68. Thus, it might have been set to 2 while running UPDATESTATE(G). If so — upon returning from UPDATESTATE(G) to RUNNINGLOOP(G) it would not pass the loop condition in Line 46, but enter the one in Line 50 and set $state_p(G) = 0$ using Line 55 before the end of the round. □

Corollary 8.2. Let p be a non-faulty node and $r \geq \iota_0 + 1$ be a global round. The condition on Line 32 will not hold for p in r.

Proof. By applying Claim 8.1 to $r - 1 \geq \iota_0$. □

Fault detection and 4-cast validity

Definition 8.5. Let r be a global round number. The system is said to be $\overline{\text{4C-Accept}}$-correct in r iff:

(1) the properties of 4-cast hold when considering the $\overline{\text{4C-Accept}}$ signals raised by non-faulty nodes in r, instead of the 4C-Accept signals.

(2) the agreement properties of 4-cast also hold when comparing the 4C-Accept and $\overline{\text{4C-Accept}}$ signals raised by some non-faulty node in r.

Lemma 8.1. Let $r \geq \iota_0 + \Delta^{4c}_{stb}$ be a global round number and assume $\lfloor \cup BAD \rfloor_{r'}$ contained only faulty nodes for every $r' \in \{r - \Delta^{4c}_{stb} + 1, \ldots, r\}$, then the system is $\overline{\text{4C-Accept}}$-correct in r.

Proof. First, note that since 4-cast requires only Δ^{4c}_{stb} rounds to stabilize, 4C-Accept (or $\overline{\text{4C-Accept}}$) signals raised in round r are affected only by the communication received during rounds $\{r - \Delta^{4c}_{stb} + 1, \ldots, r\}$.

Now, for $r' \in \{r - \Delta^{4c}_{stb} + 1, \ldots, r\}$, a non-faulty p running $\overline{\text{4C-RunningLoop}}$ will ignore the nodes in $\lfloor BAD_p \rfloor_{r'} \subseteq \lfloor \cup BAD \rfloor_{r'}$ which are all faulty. Ignoring messages from faulty nodes does not affect the properties of 4-cast, because it is equivalent to these faulty nodes not sending those messages at all.

Therefore, signals raised by $\overline{\text{4C-RunningLoop}}$ in r relate to the others like signals from another non-faulty node might have related. □

Lemma 8.2. Let p be a non-faulty node. Had p added q to BAD_p in global round r, $r \geq \iota_0 + \Delta^{4c}_{stb}$, then q is faulty.

Proof. Since $r \geq \iota_0$, q must have been added using Line 22.

Had p entered the condition on Line 21 for some node q in r, then p received a message that q 4-casted with confidence $\in \{0, 1\}$. Using the *unforgeability* and *validity* properties of 4-cast, this implies that q is faulty. □

Claim 8.2. Let $r \geq \iota_0 + \Delta_{rmv} + \Delta^{4c}_{stb} + 1$, $\lfloor \cup BAD \rfloor_r$ contains only faulty nodes.

Proof. Let p, q be two non-faulty nodes, and assume to the contrary that $q \in \lfloor BAD_p \rfloor_r$. Note that from Lemma 8.2, q must have been added to BAD_p before round $\iota_0 + \Delta^{4c}_{stb}$. Thus, $\lfloor \rho\text{-}BAD_p(q) \rfloor_r = \lfloor \rho\text{-}BAD_p(q) \rfloor_{r-1} = \lfloor \rho\text{-}BAD_p(q) \rfloor_{\iota_0 + \Delta^{4c}_{stb}}$.

Note that $\lfloor \rho\text{-}BAD_p(q) \rfloor_{\iota_0+\Delta^{4c}_{stb}} \leq lt(\iota_0+\Delta^{4c}_{stb}, p)$, or else p would have removed q from BAD_p using CLEANUP in round $\iota_0 + \Delta^{4c}_{stb}$ in contradiction. Therefore, $\lfloor \rho\text{-}BAD_p \rfloor_{r-1} = \lfloor \rho\text{-}BAD_p(q) \rfloor_{\iota_0+\Delta^{4c}_{stb}} \leq lt(\iota_0 + \Delta^{4c}_{stb}, p)$.

However, note that $lt(r-1,p) - \Delta_{rmv} \geq lt(\iota_0 + \Delta_{rmv} + \Delta^{4c}_{stb}, p) - \Delta_{rmv} = lt(\iota_0 + \Delta^{4c}_{stb}, p) \geq \lfloor \rho\text{-}BAD_p(q) \rfloor_{r-1}$ in contradiction with the fact that q was not removed from BAD_p in $r-1$.

All in all, there will be no non-faulty nodes in $\lfloor BAD_p \rfloor_r$, and by replacing p with every non-faulty node we have that there will be no non-faulty nodes in $\lfloor \cup BAD \rfloor_r$. □

Corollary 8.3. Let $r \geq \iota_0 + \Delta_{rmv} + 2\Delta^{4c}_{stb}$ be a global round number, the system is $\overline{\text{4C-ACCEPT}}$-correct in r.

Proof. Follows by using Claim 8.2 together with Lemma 8.1. □

From here on, we will denote $r_{clean} = \iota_0 + \Delta_{rmv} + 2\Delta^{4c}_{stb}$.

Properties of *joined* and *started*

Lemma 8.3. Let p be a non-faulty node and let $r \geq \iota_0$ be a global round. $\lceil joined_p(G) \rceil_r \leq lt(r,p)$ with equality **iff** p has restarted RUNNINGLOOP(G) in r.

Proof. Note that in the beginning of r CLEANUP makes sure that $joined_p(G) < lt(r,p)$. Had $joined_p(G)$ also been set during r, it must have been to $lt(r,p)$ using Line 40. □

Lemma 8.4. Let p be a non-faulty node and let $r \geq \iota_0 + 1$ be a global round.

(1) $\lceil joined_p(G) \rceil_r \geq \lfloor joined_p(G) \rfloor_r$ with inequality iff p restarted RUNNINGLOOP(G) in r;

(2) and $\lceil started_p(G) \rceil_r \neq \lfloor started_p(G) \rfloor_r$ only if p restarted RUNNINGLOOP(G) in r.

Proof. From Claim 8.1, $\lfloor joined_p(G) \rfloor_r < lt(r,p)$. Also, from Corollary 8.2, p will not change $joined_p(G)$ or $started_p(G)$ in r using CLEANUP.

Hence, has $joined_p(G)$ been changed in r it must have been on Line 40 — implying that RUNNINGLOOP(G) was restarted in r. If so, $\lceil joined_p(G) \rceil_r = lt(r,p) > \lfloor joined_p(G) \rfloor_r$ and we have the required for (1). As for (2), simply note that p changes $started_p(G)$ only before restarting RUNNINGLOOP(G). □

Corollary 8.4. Let p be a non-faulty node, let $r \geq \iota_0 + 1$ be a global round, and let $r' = \max\{rt(\lfloor joined_p(G) \rfloor_r), \iota_0 - 1\}$. p has not restarted RunningLoop(G) in any of the rounds $r' + 1, \ldots, r - 1$.

Proof. First, note that $r' + 1 \geq \iota_0$. Assume to the contrary that p has restarted RunningLoop(G) in $\hat{r} \in \{r' + 1, \ldots, r - 1\}$, then from Lemma 8.3 it had set $\lceil joined_p(G) \rceil_{\hat{r}} = lt(\hat{r}, p)$. By iteratively applying Lemma 8.4 on the last we have that $\lfloor joined_p(G) \rfloor_r \geq \lceil joined_p(G) \rceil_{\hat{r}} = lt(\hat{r}, p)$.

Thus, $rt(\lfloor joined_p(G) \rfloor_r) \geq \hat{r} > r' \geq rt(\lfloor joined_p(G) \rfloor_r)$ in contradiction! □

Corollary 8.5. Let p be a non-faulty node, let $r \geq \iota_0 + 1$ be a global round, and let $r' = rt(\lfloor joined_p(G) \rfloor_r)$. If $r' \geq \iota_0$, then p must have restarted RunningLoop(G) in r'.

Proof. First, from Corollary 8.4 we have that p has not restarted RunningLoop(G) in any of the rounds $r' + 1, \ldots, r - 1$. Combined with Corollary 8.2 we have that $\lceil joined_p(G) \rceil_{r'} = \lfloor joined_p(G) \rfloor_{r'+1} = \lfloor joined_p(G) \rfloor_r = lt(r', p)$.

By applying Lemma 8.3, we have that p must have restarted RunningLoop(G) in r'. □

Phase cleanliness

Lemma 8.5. Let p be a non-faulty node and let $r \geq \iota_0$ be a global round. Had p restarted RunningLoop(G) in r, then it would have been running RunningLoop(G) in phase with (G, r_0) in the end of r, s.t. $r_0 = r - k\Delta_{4c}$, $k \geq 1$ and $\lceil joined_p(G) \rceil_r = r$.

Proof. From Lemma 8.3 we have that $\lceil joined_p(G) \rceil_r = lt(r, p)$. Also, note that $r_0 = \lceil started_p(G) \rceil_r$ by Definition 8.1.

Had p restarted RunningLoop(G) in r using Line 8, then $\lceil started_p(G) \rceil_r = lt(r, p) - \Delta_{4c}$ as required. Had p restarted RunningLoop(G) using Line 15, then from Line 10 together with Line 14 we have that $\lceil started_p(G) \rceil_r = lt(r, p) - (k+1)\Delta_{4c}$ for $k \geq 1$ as required. □

Lemma 8.6. Let p be a non-faulty node, and let $r \geq \iota_0 + \Delta_{4c}$. If p is running RunningLoop(G) in r then, $joined_p(G) = started_p(G) + k\Delta_{4c}$ s.t. $k \geq 1$ (after executing JoinIn).

Proof. Had p restarted RunningLoop(G) in r, the required follows from Lemma 8.5. Otherwise, p must have been running RunningLoop(G) in the beginning of r.

Had p restarted RUNNINGLOOP(G) in $r' \in \{r - \Delta_{4c}, \ldots, r-1\}$, then by choosing the maximal r' and iteratively applying Lemma 8.4 to $r'+1, \ldots, r$ we have that $\lfloor joined_p(G) \rfloor_r = \lceil joined_p(G) \rceil_{r'}$ and $\lfloor started_p(G) \rfloor_r = \lceil started_p(G) \rceil_{r'}$ and the required follows by applying Lemma 8.5 to r'.

Otherwise, since the only blocking line in RUNNINGLOOP(G) is Line 47 which blocks for Δ_{4c} rounds, we have that p must have executed UPDATESTATE(G) in some $r' \in \{r - \Delta_{4c}, \ldots, r-1\}$. Had p passed the condition in Line 59 in r', it would have stopped RUNNINGLOOP(G), in contradiction with the fact that it is still running in the beginning of r. Therefore, by iteratively applying Lemma 8.4 to $r'+1, \ldots, r$ and from the fact that the said condition did not hold for p in r' we have the required. □

Lemma 8.7. Let p be a non-faulty node, and let $r \geq \iota_0 + \Delta_{4c}$ be a global round. Has p been running RUNNINGLOOP(G) in phase with (G, r_0) in the beginning of r, then it had been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of $r - \Delta_{4c}$.

Proof. From Definition 8.1 we have that $\lfloor started_p(G) \rfloor_r = r_0$ and that $r = r_0 + k_1 \Delta_{4c}$, $k_1 \geq 1$.

From Lemma 8.6 we have that $rt(\lfloor joined_p(G) \rfloor_r) = r_0 + k_2 \Delta_{4c}$, $k_2 \geq 1$. Moreover, from Claim 8.1 we have that $rt(\lfloor joined_p(G) \rfloor_r) < r$ and therefore $k_2 < k_1 \Rightarrow k_1 \geq 2$, and $rt(\lfloor joined_p(G) \rfloor_r) \leq r - \Delta_{4c}$.

Applying Corollary 8.4, we have that p has not have restarted RUNNINGLOOP(G) in $r - \Delta_{4c} + 1, \ldots, r-1$. Hence, p has been running RUNNINGLOOP(G) in the end of $r - \Delta_{4c}$, and by iteratively applying Lemma 8.4 to $r - \Delta_{4c} + 1, \ldots, r-1$ we have that $\lceil started_p(G) \rceil_{r-\Delta_{4c}} = \lfloor started_p(G) \rfloor_r = r_0$. That together with the fact that $r - \Delta_{4c} = r_0 + (k_1 - 1)\Delta_{4c}$ and $k_1 - 1 \geq 1$ implies that p has been in phase with (G, r_0) in the end of $r - \Delta_{4c}$. □

Claim 8.3. Let p be a non-faulty node, and let $r \geq \iota_0 + \Delta_{4c}$. p has been running RUNNINGLOOP(G) in phase with (G, r_0) in r, and $r \geq r_0 + 2\Delta_{4c}$ **iff** p has executed UPDATESTATE(G) in r.

Proof. First we will prove the reversed direction, and note that UPDATESTATE(G) is executed only from within RUNNINGLOOP(G).

Had p restarted RUNNINGLOOP(G) in r using Line 8, it would have had $\lceil started_p(G) \rceil_r = lt(r, p) - \Delta_{4c}$ and could not have passed Line 43 in order to execute UPDATESTATE(G) in r, in contradiction to the assumption. Thus, either p has been

restarted RunningLoop(G) in r using Line 15 or it has been running RunningLoop(G) in phase with (G, r_0) in the beginning of r.

Had p restarted RunningLoop(G) in r using Line 15, then from Lemma 8.5 RunningLoop(G) would have been running in phase with (G, r_0), and from Corollary 8.1 it would have had $r_0 \leq r - 2\Delta_{4c}$.

Had p been running RunningLoop(G) in the beginning of r, then since it has executed UpdateState(G) in r, it could not have restarted RunningLoop(G) or executed Line 47 in $r - \Delta_{4c} + 1, \ldots, r - 1$ (or else it would have been blocking in r). Therefore, p must have been running RunningLoop(G) in the end of $r - \Delta_{4c}$ and executed Line 47 then. Since p has not stopped RunningLoop(G), we have that the condition in Line 59 has been false in $r - \Delta_{4c}$. Combined with Definition 8.1 and since $r - \Delta_{4c} \geq \iota_0$, this means that p has been running RunningLoop(G) in phase with (G, r_0) in $r - \Delta_{4c} \geq r_0 + \Delta_{4c}$.

Since we have shown that p has not restarted RunningLoop(G) in $r - \Delta_{4c} + 1, \ldots, r-1$, by iteratively applying Lemma 8.4, p must still be running RunningLoop(G) in phase with (G, r_0) in r, and $r \geq r_0 + 2\Delta_{4c}$.

Now for the forward direction.

First, had p restarted RunningLoop(G) in r, since $r \geq r_0 + 2\Delta_{4c}$ it would have passed the condition in Line 43 and executed UpdateState(G) in phase with (G, r_0) in r.

Had p not restarted RunningLoop(G) in r, it must have been running it in phase with (G, r_0) in the beginning of r.

Combining with Lemma 8.5 and Lemma 8.4, we have that that p could not have restarted RunningLoop(G) in any $r' \in \{r - \Delta_{4c} + 1, \ldots, r - 1\}$, or else it would have had $rt(\lfloor started_p(G) \rfloor_r) = r' - k\Delta_{4c} \neq r - k\Delta_{4c}$ in contradiction. Thus, p has been running RunningLoop(G) in phase with (G, r_0) in the end of $r - \Delta_{4c}$ as well. By iteratively applying Lemma 8.4 to $r' \in \{r - \Delta_{4c}, \ldots, r - 1\}$ we have that $\lceil started_p(G) \rceil_{r'} = \lceil started_p(G) \rceil_r = r_0 = r - k\Delta_{4c}$.

Therefore, had p executed UpdateState(G) in any $r' \in \{r - \Delta_{4c} + 1, \ldots, r - 1\}$, the condition in Line 59 would have been false and RunningLoop(G) would have stopped in contradiction to the assumption that it is still running in the beginning of r.

Since the only blocking line in RunningLoop(G) is Line 47 which blocks for Δ_{4c} rounds, we have that p cannot still be blocking in r and so will execute UpdateState(G) in phase with (G, r_0) in r. □

Corollary 8.6. Let p be a non-faulty node, and let $r \geq \iota_0 + \Delta_{4c}$ be a global round. Had p executed UPDATESTATE(G) in r, the condition on Line 59 must have held for p in r.

Proof. Note that from Claim 8.3, had p executed UPDATESTATE(G) in r, it must have been in phase with (G, r_0) s.t. $r \geq r_0 + 2\Delta_{4c}$.

Also, from Lemma 8.6 and Lemma 8.3 we have that $r \geq joined_p(G) = started_p(G) + k\Delta_{4c}$ with $k \geq 1$ when executing Line 59.

It follows that the condition in Line 59 will be false. □

Corollary 8.7. The conditions in Line 32 and Line 59 will not hold for any non-faulty node in rounds $\geq \iota_0 + \Delta_{4c}$.

Proof. Follows from Corollary 8.2 and Corollary 8.6. □

Therefore, when regarding the stopping of RUNNINGLOOP after round $\iota_0 + \Delta_{4c}$ in the following proofs we can assume that RUNNINGLOOP is stopped only by returning.

Claim 8.4. A non-faulty node p has 4-casted an A-*share* message in phase with (G, r_0) in global round r, $r \geq \iota_0$ **iff** one of the following conditions has held:

(1) p has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r;

(2) or p has returned successfully from RUNNINGLOOP(G) in r after running UPDATESTATE(G) in phase with (G, r_0).

Proof. First, it follows from the algorithm that in $r \geq \iota_0$ a non-faulty node p 4-casts only A-*share* messages that are in phase with $(G, rt(\lceil started_p(G) \rceil_r))$ and only using Lines 47 and 51.

In both cases, p restarted RUNNINGLOOP(G) or executed UPDATESTATE(G) in round r prior to 4-casting. Had p restarted RUNNINGLOOP(G) then it is in phase from Lemma 8.5. Had it executed UPDATESTATE(G), it is in phase from Claim 8.3.

The required follows immediately from the fact that Line 47 is executed iff p has been running RUNNINGLOOP(G) in phase in the end of r, and that Line 51 is executed iff p has returned successfully in r. □

Algorithm logic cleanliness

Lemma 8.8. Let p be a non-faulty node and let $r \geq r_{clean}$. The condition in Line 10 will not be satisfied for more than one (G, r_0, m) triplet in r.

Proof. First, from Corollary 8.3 and the fact that $r \geq r_{clean}$, we have that the received 4-casts in round r satisfy the properties of 4-CAST. Second, note that from the *uniqueness* property of 4-CAST, p will not raise more than one $\overline{\text{4C-ACCEPT}}$ signal per node in each round.

Therefore, and since $n \geq 3t + 1$, $n - t$ must be a majority and so p cannot have the condition in Line 10 satisfied for more than one (G, r_0, m) triplet. □

Claim 8.5. Let p be a non-faulty node, let r_0 be a global round, and let $r \geq \max\{r_0 + 2\Delta_{4c}, r_{clean}\}$. The condition in Line 10 has held for p with A-*share* 4-casts in phase with (G, r_0) and with value $\lceil v_p(G) \rceil_r$ in r **iff** one of the following has held:

(1) p has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r;

(2) or p has returned successfully from RUNNINGLOOP(G) in r after running UPDATESTATE(G) in phase with (G, r_0).

Proof. First, we will prove the forward direction: assume the condition in Line 10 held for p with the triplet (G, r_0, m) and we will show that it will be running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r and that $\lceil v_p(G) \rceil_r = m$.

Note that from Lemma 8.8, the said condition did not hold for p with any other triplet in r.

Had p not been running RUNNINGLOOP(G) in phase with (G, r_0) in the beginning of r, then from Corollary 8.1 it will not pass the condition in Line 11 but execute Line 15 and restart RUNNINGLOOP(G) to be running in phase with (G, r_0) in the end of r.

Otherwise, from Corollary 8.1, p will pass the condition in Line 11 and set $timeout_p(G) = lt(r, p)$. Therefore, either p will not break from the loop when executing RUNNINGLOOP(G) and be running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r **or** p has set $state_p(G) = 2$ in UPDATESTATE(G) in which case it will return successfully in r after running UPDATESTATE(G) in phase with (G, r_0).

In any case it follows from Corollary 8.1 that $r \geq r_0 + 2\Delta_{4c}$. Therefore, it follows from Claim 8.3 that p will execute UPDATESTATE(G) in r. Thus, p will make sure

that $\lceil v_p(G) \rceil_{=} m$ when going past Line 63, since $n \geq 3t+1$ and so the $n-t$ 4-casts received with the same value m must be a majority.

As for the reversed direction: assume p has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of $r \geq r_0$ or has returned successfully from RUNNINGLOOP(G) after running UPDATESTATE(G) in phase with (G, r_0), then the condition in Line 10 must have held for p with the triplet (G, r_0, m).

Note that it is not possible that p restarted RUNNINGLOOP(G) in r using Line 8, because then it would be in phase with $(G, r - \Delta_{4c})$, and $r - \Delta_{4c} > r_0$ (from Corollary 8.1).

Had p not been running RUNNINGLOOP(G) in phase with (G, r_0) in the beginning of r, then it must have restarted RUNNINGLOOP(G) in r using Line 15. Hence, the condition in Line 10 must have held for messages in phase with (G, r_0).

Otherwise, p has been running RUNNINGLOOP(G) in phase with (G, r_0) in the beginning of r. By applying Claim 8.1 to $r - 1 \geq \iota_0$ we have that $\lfloor timeout_p(G) \rfloor_r < lt(r, p)$. In the case p has not returned successfully — from the fact that p is still running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r, we have that p must have reset $timeout_p(G)$ using Line 12 and so the condition in Line 10 must have held for messages in phase with (G, r_0) as well. Had p returned successfully — note that the condition in Line 67 must have been true, and that the condition in Line 10 is contained in that of Line 67.

It also follows from Claim 8.3 that p has executed UPDATESTATE(G) in r and set $\lceil v_p(G) \rceil_r$ to the value received in the majority of the in phase A-*share* 4-casts. From Lemma 8.8 this must have been the value m of the said 4-casts. □

Lemma 8.9. Let p be a non-faulty node, and let $r \geq r_{clean} + \Delta_{4c}$ be a global round. Had p returned successfully from RUNNINGLOOP(G) in r after running UPDATESTATE(G) in phase with (G, r_0), it must have been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of $r - \Delta_{4c}$ and had $\lceil state_p(G) \rceil_{r-\Delta_{4c}} = 1$.

Proof. Note that from Claim 8.1, $\lfloor state_p(G) \rfloor_r \in \{0, 1\}$. Since p has returned successfully in r, it must have had $state_p(G) = 2$ when executing the conditional in Line 50. The only possibility is that p has executed Line 68 in r and had $state_p(G) = 1$ when starting to execute UPDATESTATE(G).

Had p restarted RUNNINGLOOP(G) in r, it would have set $state_p(G) = 0$ using Line 42 before executing UPDATESTATE(G), in contradiction. Therefore, p must have been running RUNNINGLOOP(G) in phase with (G, r_0) in the beginning of r.

Therefore, from Lemma 8.7 we have that p has been running RunningLoop(G) in phase with (G, r_0) in the end of $r - \Delta_{4c}$ as well. Moreover, $\lceil state_p(G) \rceil_{r-\Delta_{4c}} = \lfloor state_p(G) \rfloor_r = 1$ as required. □

Agreement once all are participating

Lemma 8.10. Let p be a non-faulty node. Had every non-faulty was running RunningLoop(G) in phase with (G, r_0) in the end of global round r^*, $r^* \geq r_{clean}$, then p will not restart RunningLoop(G) on any round $r > r^*$ unless either:

(1) p has returned from RunningLoop(G) unsuccessfully on $\bar{r}$ s.t. $r^* < \bar{r} < r$;

(2) p has returned from RunningLoop(G) successfully on $\bar{r}$ s.t. $r^* < \bar{r} < r - \Delta_{4c}$;

(3) another non-faulty node has returned from RunningLoop(G) unsuccessfully on $\bar{r}$ s.t. $r^* < \bar{r} < r - \Delta_{4c}$;

(4) or another non-faulty node has returned from RunningLoop(G) successfully on $\bar{r}$ s.t. $r^* < \bar{r} < r - 2\Delta_{4c}$.

Proof. First, note that from Corollary 8.7, no non-faulty will stop RunningLoop(G) on any round $> r^*$ without properly returning. Also, by applying Corollary 8.3 we know that $\overline{\text{4C-Accept}}$ messages raised on r^* or later satisfy the properties of 4-cast and we use this throughout this proof.

Now — note that p will not restart RunningLoop(G) using Line 8 until it has returned from it, because of the condition in Line 5. Had p returned successfully it will also ignore $\{\overline{\text{4C-Accept}}, G, \langle \text{A-}init, * \rangle, 2\}$ signals for the next Δ_{4c} rounds. This gives us the necessary condition [(1) or (2)] for restarting using Line 8.

We will now show that [(1), (2), (3) or (4)] is a necessary condition for restarting using Line 15.

First, note that p will not be able to restart RunningLoop(G) using Line 15 to be in phase with (G, r_0) until it has returned from it, because the condition in Line 13 will stay false. Assuming p has returned from RunningLoop(G) in some round $\bar{r}$: had it returned unsuccessfully then condition (1) will hold in $r \geq \bar{r} + \Delta_{4c}$. Had p returned successfully in $\bar{r}$ it will not be "pulled in" before $\bar{r} + 3\Delta_{4c}$ because it will ignore the relevant share messages (from Line 52) and so condition (2) will hold in $r \geq \bar{r} + 3\Delta_{4c}$.

For restarting RunningLoop(G) with different phases, p must receive appropriate A-*share* 4-casts from at least $n - 2t$ non-faulty nodes. Note that we assumed that by

the end of round r^* all non-faulty nodes are running RUNNINGLOOP(G) with the same phase. Hence, other non-faulty nodes have to restart RUNNINGLOOP(G) with the new phase at least Δ_{4c} rounds prior to p's restart for them to "pull" p with the condition on Line 13.

Let q be the first non-faulty node to restart with a different phase after round r^*. From the previous paragraph, as q is the first — it could not have been "pulled out" and had to restart using Line 8. From the first paragraph, this means that q must have previously returned unsuccessfully from RUNNINGLOOP(G) in round $\bar{r}$ s.t. $\bar{r} < r - \Delta_{4c}$, or returned unsuccessfully in round $\bar{r}$ s.t. $\bar{<}(r - \Delta_{4c}) - \Delta_{4c}$. This completes the necessary condition [(1), (2), (3) or (4)] for restarting using Line 15. □

Claim 8.6. Let $r \geq r_{clean}$ and m, s.t.:

(1) Every non-faulty node has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r;

(2) and every non-faulty node has had $\lceil v(G) \rceil_r = m$.

Then, every non-faulty node q will raise $\{\text{A-ACCEPT}, G, m, \rho_0^q\}$ s.t. $rt(\rho_0^q) = r_0$ all within the round interval $[r + \Delta_{4c},\ r + 2\Delta_{4c}]_{\Delta_{4c}}$.

Proof. First, note that from Claim 8.4 every non-faulty node will 4-cast an A-*share* in phase with (G, r_0) in round r, all with the value m. From Corollary 8.3 and the *validity* property of 4-CAST, in round $r + \Delta_{4c}$ every non-faulty node will receive at least $n - t$ A-*share* 4-casts in phase with (G, r_0) with the value m and with confidence 2.

From Definition 8.1 we have that $r \geq r_0 + \Delta_{4c}$. Thus, $r + \Delta_{4c} \geq r_0 + 2\Delta_{4c}$ and we can use Claim 8.5 to see that every non-faulty node will have $\lceil v(G) \rceil_{r+\Delta_{4c}} = m$, some will be running RUNNINGLOOP(G) in phase with (G, r_0) in the end of $r + \Delta_{4c}$ and some might have returned successfully after running UPDATESTATE(G) in phase with (G, r_0). We will denote the first set of nodes as S_1 and the second as S_2.

Note that:

(1) Before successfully returning, the nodes in S_2 must have raised the signal $\{\text{A-ACCEPT}, G, v(G), started_p(G)\}$. Moreover, from Definition 8.1 we have that $rt(started(G)) = r_0$ as required, and from the previous paragraph we have that $v(G) = m$.

(2) While running UpdateState(G), the nodes in S_1 must have passed the condition in Line 68. Combined with Claim 8.1 they have $\lceil state(G) \rceil_{r+\Delta_{4c}} = 1$ (had it been 2 they would be in S_2).

From Claim 8.4 again, in the end of round $r + \Delta_{4c}$ both the nodes in S_1 and the nodes in S_2 will 4-cast A-*share* messages in phase with (G, r_0) and the value m.

Moving on to round $r + 2\Delta_{4c}$ — note that from Lemma 8.10, no non-faulty node in S_1 will restart RunningLoop(G) in $r + \Delta_{4c} + 1, \ldots, r + 2\Delta_{4c}$.

As in $r + \Delta_{4c}$, in round $r + 2\Delta_{4c}$ the nodes in S_1 will again receive at least $n - t$ 4-casts for A-*share* messages in phase with (G, r_0) and with value m and confidence 2.

Every node $p \in S_1$ will then pass the condition in Line 68, set $state_p(G) = 2$, break from the loop in RunningLoop(G), pass the condition in Line 50 and raise $\{\text{A-Accept}, G, v_p(G), started_p(G)\}$. As before, from Definition 8.1, we have that $rt(started_p(G)) = r_0$ as required, and from Claim 8.5 we have that $v_p(G) = m$.

Combining all the above, we have shown that every non-faulty node q will raise $\{\text{A-Accept}, G, m, \rho_0^q\}$ s.t. $rt(\rho_0^q) = r_0$, all within the round interval $[r + \Delta_{4c}, r + 2\Delta_{4c}]_{\Delta_{4c}}$. □

Decay

Lemma 8.11. Let p be a non-faulty node, and let $r \geq r_{clean}$ be a global round. Had the condition in Line 10 held for p in r with messages in phase with (G, r_0), then at least $n - t - f + |\lfloor BAD_p \rfloor_r|$ of these 4-casts were from non-faulty nodes.

Proof. First, note that from Corollary 8.1, p has received in r at least $n - t$ 4-casts A-*share* messages in phase with (G, r_0) and with confidence ≥ 1.

From Corollary 8.3 and the fact that $r \geq r_{clean}$, only non-faulty nodes are in $\lfloor BAD_p \rfloor_r$, and note that $\overline{\text{4-cast}}$ does not raise $\overline{\text{4C-Accept}}$ signals in r for nodes in $\lfloor BAD_p \rfloor_r$. That, together with the fact that $n \geq 3t + 1$, means that at least $n - t - f + |\lfloor BAD_p \rfloor_r|$ of the received 4-casts must be from non-faulty nodes. □

Corollary 8.8. Let r_0 be a global round, let $r \geq \max\{r_0 + 2\Delta_{4c}, r_{clean}\}$, and let $n_{BAD} = |\lfloor \cap BAD \rfloor_r|$. If less than $n - t - f + n_{BAD}$ non-faulty nodes have 4-casted A-*share* messages in phase with (G, r_0) in $r - \Delta_{4c}$, then no non-faulty node will run RunningLoop(G) in phase with (G, r_0) in any round $r' > r$ nor in the end of r.

Proof. By applying Claim 8.5 and then Lemma 8.11 to all non-faulty nodes in r, we have that no non-faulty node will be running RunningLoop(G) in phase with (G, r_0)

in the end of r. By iteratively applying these lemmata to the rounds in $[r + \Delta_{4c}, r']_{\Delta_{4c}}$ and using Lemma 8.7 we have the required. □

Corollary 8.9. Let $r \geq r_{clean}$ and m, s.t.:

(1) Every non-faulty node has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r;

(2) and every non-faulty node has had $\lceil v(G) \rceil_r = m$.

No non-faulty node q will be running RUNNINGLOOP(G) in phase with (G, r_0) in any round $r' > r + 2\Delta_{4c}$.

Proof. It follows from Claim 8.6 that all non-faulty nodes will return successfully from RUNNINGLOOP(G) in phase with (G, r_0) in the round interval $[r + \Delta_{4c}, r + 2\Delta_{4c}]_{\Delta_{4c}}$. The proof also implies that no non-faulty node will be running RUNNINGLOOP(G) in phase with (G, r_0) in the end of round $r + 2\Delta_{4c}$.

Moreover, because of Line 52, we know that no non-faulty node will be able to restart RUNNINGLOOP(G) in phase with (G, r_0) in rounds $[r + 2\Delta_{4c}, r + 3\Delta_{4c}]$. From Claim 8.4, we have that no non-faulty node will 4-cast an A-*share* message in phase with (G, r_0) in round $r + 3\Delta_{4c}$.

By applying Corollary 8.8 to round $r + 4\Delta_{4c}$ and using the fact that no non-faulty node will be running RUNNINGLOOP(G) in the beginning of $r + 4\Delta_{4c}$ (from Lemma 8.7), we have the required. □

Claim 8.7. Let p be a non-faulty node, let r_0 be a global round, and let $r \geq \max\{r_0 + 2\Delta_{4c}, r_{clean}\}$. Had p been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of global round r, then either the following two statements hold:

(1) Every non-faulty node will be running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r;

(2) and every non-faulty node q will have $\lceil v_q(G) \rceil_r = \lceil v_p(G) \rceil_r$.

or there will be some $q \in \mathcal{P}$ s.t. $q \notin \lfloor BAD_p \rfloor_r$ and $q \in \lceil BAD_p \rceil_r$.

Proof. First, from Claim 8.5 we have that the condition in Line 10 held for p with messages in phase with (G, r_0) in r. Using Corollary 8.1 we have that p has received at least $n - t$ A-*share* 4-casts in phase with (G, r_0) and all with the same value m and confidence ≥ 1.

Had all other non-faulty nodes received exactly the same 4-casts (same G, m and age) with confidences ≥ 1 as well, then from Claim 8.5 they will all be running RunningLoop(G) in phase with (G, r_0) in the end of r as well.

Moreover, from Claim 8.3 every non-faulty node will execute UpdateState(G) in r. Thus, every non-faulty node q will set $v_p(G)$ to the same value m when executing Line 64 in UpdateState(G), as $n \geq 3t+1$ and so the $n-t$ 4-casts received with the same value m must be a majority.

Otherwise — if not all non-faulty nodes received these $n-t$ 4-casts with confidences ≥ 1 — at least one non-faulty node $\hat{p} \neq p$ has received one of the said 4-casts with either: (1) a different value; (2) confidence < 1; or (3) not raised it at all. Assume w.l.o.g that this "different" signal was for a 4-cast invoked by q.

Note that $q \notin \lfloor BAD_p \rfloor_r$ or else p would not have raised the $\overline{\text{4C-Accept}}$ signal about q in the first place. Since $r \geq r_{clean}$ and by applying Corollary 8.3, we know that the properties of 4-cast hold between the $\overline{\text{4C-Accept}}$ and 4C-Accept signals for all non-faulty nodes.

That is, as p is non-faulty and has raised the signal about q with confidence ≥ 1, we know that $\hat{p}$ must have raised a signal about the 4-cast from q as well. Therefore, $\hat{p}$ has raised the signal about the 4-cast from q either with confidence < 1 or with a different value (which implies confidence < 1 as well, from the *value agreement* property of 4-cast). Therefore, from the *confidence agreement* properties of 4-cast, we know that p has raised $\overline{\text{4-cast}}$ on r with confidence 1 and not 2.

All in all, p will add q to BAD_p when executing Snitch in r. That is — $q \notin \lfloor BAD_p \rfloor_r$ but $q \in \lceil BAD_p \rceil_r$ as required. □

Corollary 8.10. Let p be a non-faulty node, let r_0 be a global round, and let $r \geq \max\{r_0 + 2\Delta_{4c}, r_{clean}\}$. Also, let $r' = \max\{r - \Delta_{rmv} + 1, r_0 + 2\Delta_{4c}, r_{clean}\}$. Had p been running RunningLoop(G) in phase with (G, r_0) in the end of $f+1$ rounds in the round interval $[r', r]$ then for some $\hat{r} \in [r', r]$ we had that:

(1) Every non-faulty node was running RunningLoop(G) in phase with (G, r_0) in the end of $\hat{r}$;

(2) and every non-faulty node q had $\lceil v_q(G) \rceil_{\hat{r}} = \lceil v_p(G) \rceil_{\hat{r}}$.

Proof. Assume to the contrary that the claim is incorrect. From Claim 8.7 we have that a "new" faulty node must have been added to BAD_p in each one of the rounds in which p has been running in the end of (as $r' \geq r_{clean}$).

Therefore, and since $r - r' < \Delta_{rmv}$, we have that $\lfloor BAD_p \rfloor_r$ contained at least $f+1$ nodes in contradiction to Claim 8.2. □

Corollary 8.11. Let $r \geq r_{clean}$, let p be a non-faulty node, and let $r' = \max\{r - \Delta_{rmv} + 1, r_{clean}\}$. Had p 4-casted A-*share* messages that were in phase with (G, r_0) in $2f+2$ rounds in the round interval $[r'+1, r]_{\Delta_{4c}}$ then for some $\hat{r} \in [r', r]_{\Delta_{4c}}$ we had that:

(1) Every non-faulty node was running RUNNINGLOOP(G) in phase with (G, r_0) in the end of $\hat{r}$;

(2) and every non-faulty node q had $\lceil v_q(G) \rceil_{\hat{r}} = \lceil v_p(G) \rceil_{\hat{r}}$.

Proof. Had p 4-casted an A-*share* message in phase with (G, r_0) in $\hat{r} \in [r'+1, r]_{\Delta_{4c}}$, we deduce from Claim 8.4 that in the end of $\hat{r}$ p has either been running RUNNINGLOOP(G) in phase with (G, r_0) or it had returned successfully from RUNNINGLOOP(G) after running UPDATESTATE(G) in phase with (G, r_0). For the second case, by applying Lemma 8.9 to $\hat{r}$ we have that p has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of $\hat{r} - 1 \in [r', r]_{\Delta_{4c}}$.

Also, from Claim 8.4 we have that in every round in the end of which p has been running RUNNINGLOOP(G) in phase with (G, r_0), p had 4-casted an A-*share* message in phase with (G, r_0).

Therefore, for every 4-cast that originated from a successful return we have at least one 4-cast that originated from running in phase in the end of a round. Hence, p has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of at least $f+1$ rounds in $[r', r]_{\Delta_{4c}}$.

By applying Corollary 8.10 we have the required. □

Claim 8.8 (Decay)**.** In global round $r \geq r_{clean} + (4f+5)\Delta_{4c}$ no non-faulty node will be running RUNNINGLOOP(G) in phase with (G, r_0) s.t. $r_0 < r - (4f+5)\Delta_{4c}$.

Proof. Assume to the contrary that in $r \geq r_{clean} + (4f+5)\Delta_{4c}$ some non-faulty node p has been running RUNNINGLOOP(G) in phase with (G, r_0) s.t. $r_0 < r - (4f+5)$.

If on any round $r' \in \{r - (4f+5)\Delta_{4c}, \ldots, r - 3\Delta_{4c}\}$ it was true that:

(1) Every non-faulty node has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r';

(2) and every non-faulty node has had $\lceil v(G) \rceil_{r'} = m$.

Then using Corollary 8.9, p could not have been running RunningLoop(G) in r in contradiction to the assumption.

Also, from Corollary 8.8 at least $n-t-f$ non-faulty nodes have 4-casted A-*share* messages in phase with (G, r_0) in the end of every round in $[r-(4f+5)\Delta_{4c}, r-2\Delta_{4c}]_{\Delta_{4c}}$ or else p could not have been running RunningLoop(G) in the end of r in contradiction to the assumption.

Combining the above with Corollary 8.11, and since we will require that $\Delta_{rmv} > (4f+4)\Delta_{4c}$, no non-faulty node could have sent A-*share* messages in phase with (G, r_0) in the end of $2f+2$ rounds in the interval $[r-(4f+5)\Delta_{4c}, r-3\Delta_{4c}]_{\Delta_{4c}}$. Otherwise, p could not have been running RunningLoop(G) in phase with (G, r_0) in r, in contradiction to the assumption.

Therefore, since there are $n-f$ non-faulty nodes, each have not sent more than $2f+1$ messages with the said phase in the said interval, and in every round in the interval at least $n-t-f$ non faulty nodes must have sent messages, we have that the length of the interval is at most:

$$\begin{aligned}
\frac{(n-f)(2f+1)}{n-t-f} &= \frac{(n-t-f+t)(2f+1)}{n-t-f} \\
&= (2f+1) + 2\frac{t\cdot f}{n-t-f} + \frac{t}{n-t-f} \\
&\leq (2f+1) + 2\frac{t\cdot f}{t+1} + \frac{t}{t+1} \\
&\leq (2f+1) + 2\frac{(t+1)\cdot f}{t+1} + 1 \\
&= 4f+2
\end{aligned}$$

Where the first inequality follows from the fact that $n \geq 3t+1$.

However:

$$\begin{aligned}
|[r-(4f+5)\Delta_{4c}, r-3\Delta_{4c}]_{\Delta_{4c}}| &= \frac{(r-3\Delta_{4c}) - (r-(4f+5)\Delta_{4c})}{\Delta_{4c}} + 1 \\
&= 4f+2+1 \\
&> 4f+2
\end{aligned}$$

in a contradiction! □

Termination

Claim 8.9 (Termination)**.** A non-faulty node p that is running RUNNINGLOOP(G) in the end of global round r, $r \geq r_{clean}+(f+3)\Delta_{4c}$, has $rt(\lceil joined_p(G)\rceil_r) \geq r-(f+3)\Delta_{4c}$.

Proof. First, let $r' = rt(\lceil joined_p(G)\rceil_r)$. Assume to the contrary that $r' < r-(f+3)\Delta_{4c}$, and note that $r-(f+3)\Delta_{4c} \geq r_{clean}$.

By applying Corollary 8.4, we have that p has not restarted RUNNINGLOOP(G) in $r-(f+3)\Delta_{4c}, \ldots, r$ where all of these rounds are $\geq r_{clean}$. Since p is still running RUNNINGLOOP(G) in the end of r, we also know that p must have been running RUNNINGLOOP(G) in the end of all of the said rounds.

From Corollary 8.9, if on any round $r' \in \{r-(f+3)\Delta_{4c}, \ldots, r-3\Delta_{4c}\}$ it was true that:

(1) Every non-faulty node has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r';

(2) and every non-faulty node has had $\lceil v(G)\rceil_{r'} = m$.

p could not have been running RUNNINGLOOP(G) in the end of r in contradiction to the assumption.

However, as said — p has been running in the end of the $f+1$ rounds $\{r-(f+3)\Delta_{4c}, \ldots, r-3\Delta_{4c}\}$. Since $r-(f+3)\Delta_{4c} \geq r-\Delta_{rmv}+1$ and $r-(f+3)\Delta_{4c} \geq r_{clean}$ it follows from Corollary 8.10 in contradiction to the assumption, that in one of these rounds the conditions presented in the previous paragraph held. □

Agreement

Lemma 8.12. Let p be the first non-faulty node to set $state_p(G) = 1$ when running UPDATESTATE(G) in phase with (G, r_0) s.t. $r_0 \geq r_{clean}$ in global round r, $r \geq \iota_0$, then:

(1) $r \geq r_0 + 2\Delta_{4c}$;

(2) every non-faulty node will be running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r;

(3) and every non-faulty node q will have $\lceil v_q(G)\rceil_r = \lceil v_p(G)\rceil_r$.

Proof. Note that for p to set $state_p(G) = 1$ it had to execute Line 68 and pass the condition in Line 67. Also, we may note that the condition in Line 10 is contained in that of Line 67 and so from Claim 8.5 we have that $r \geq 2\Delta_{4c}$.

As for (2) and (3): p has set $state_p(G) = 1$ using Line 68. Therefore, it has received at least $n - t$ 4-casts of A-*share* messages in phase with (G, r_0), all with the same value v and with confidence 2.

Let q be a non-faulty node. Since $r \geq r_{clean}$ we can apply Corollary 8.3 to r, and thus from the *agreement* properties of 4-CAST, q must have raised the exact same signals as well with confidence ≥ 1. Note that it is not possible that q has ignored any of these signals by executing Line 52 in phase with (G, r_0) in a previous round, because then from Lemma 8.9 q must have set $state_q = 1$ prior to p in contradiction with p being the first.

Therefore, the condition in Line 10 will hold for every non-faulty node with the triplet (G, r_0, m). Also, note that since no non-faulty node has had $\lfloor state_p(G) \rfloor_r = 1$, it follows from Lemma 8.9 that no non-faulty node will return successfully from RUNNINGLOOP(G) in r.

As $r \geq r_0 + 2\Delta_{4c}$ we can use Claim 8.5 and the required follows. □

Corollary 8.12. Let p be the first non-faulty node to set $state_p(G) = 1$ when running UPDATESTATE(G) in phase with (G, r_0) s.t. $r_0 \geq r_{clean}$ in global round r, $r \geq \iota_0$, then every non-faulty node q will raise $\{\text{A-ACCEPT}, G, m, \rho_0^q\}$ s.t. $rt(\rho_0^q) = r_0$ all within the round interval $[r + \Delta_{4c},\ r + 2\Delta_{4c}]_{\Delta_{4c}}$.

Proof. From Lemma 8.12 we have that:

(1) $r \geq r_0 + 2\Delta_{4c} \geq r_{clean}$;

(2) every non-faulty node will be running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r;

(3) and every non-faulty node q will have $\lceil v_q(G) \rceil_r = \lceil v_p(G) \rceil_r$.

Therefore, we can use Claim 8.6 and the required follows. □

Corollary 8.13. Let p be the first non-faulty node to set $state_p(G) = 1$ when running UPDATESTATE(G) in phase with (G, r_0) s.t. $r_0 \geq r_{clean}$ in global round r, $r \geq \iota_0$, then no non-faulty node q will be running RUNNINGLOOP(G) in phase with (G, r_0) on any round $r' > r + 2\Delta_{4c}$.

Proof. Follows by applying Lemma 8.12 and then Corollary 8.9. □

Corollary 8.14. Let p be a non-faulty node, and let $r \geq \iota_0$. Had p raised a signal $\{\text{A-ACCEPT}, G, m, \rho_0^p\}$ s.t. $rt(\rho_0^p) \geq r_{clean}$ in r, it will not raise another signal of type $\{\text{A-ACCEPT}, G, *, \rho_0^p\}$ on any later round.

Proof. First, let $r_0 = rt(\rho_0^p)$. From Definition 8.1 we have that $r \geq r_{clean} + \Delta_{4c}$. Also, from Lemma 8.9 we have that p has been running RUNNINGLOOP(G) in phase with (G, r_0) in $r - \Delta_{4c}$ and had $\lceil state_p(G) \rceil_{r-\Delta_{4c}} = 1$.

Therefore, some non-faulty node has been the first to set $state_p(G) = 1$ while running UPDATESTATE(G) in phase with (G, r_0) in round r' s.t. $r_{clean} + \Delta_{4c} \leq r' \leq r - \Delta_{4c}$.

Using Corollary 8.13, we have that p will not be running RUNNINGLOOP(G) in phase with (G, r_0) (required to raise the said signal) on any round $> r + \Delta_{4c}$. As for round $r + \Delta_{4c}$, as p returned successfully in r we have from Lemma 8.10 that it will not restart with the same phase in $r + \Delta_{4c}$. □

Claim 8.10 (Agreement)**.** Had a non-faulty node p raised $\{\text{A-ACCEPT}, G, m, \rho_0^p\}$ in global round r, $r \geq r_{clean} + (4t + 8)\Delta_{4c}$, then every non-faulty node q has raised $\{\text{A-ACCEPT}, G, m, \rho_0^q\}$ within the global round interval $[r - \Phi, r - \Phi + \Delta_{4c}]$ (for some $\Phi \in [0, \Delta_{4c}]_{\Delta_{4c}}$), s.t. $rt(\rho_0^q) = rt(\rho_0^p)$.

Proof. From Lemma 8.9 we have that p has been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of $r - \Delta_{4c} \geq r_{clean} + (4t + 7)\Delta_{4c}$ with $\lceil state_p(G) \rceil_{r-\Delta_{4c}} = 1$. Putting in Claim 8.8, we have that $r_0 \geq r_{clean}$.

Therefore, and from Definition 8.1, some non-faulty node has been the first to set $state_p(G) = 1$ while running UPDATESTATE(G) in phase with (G, r_0) in round r' s.t. $r_{clean} + \Delta_{4c} \leq r' \leq r - \Delta_{4c}$. From Corollary 8.12, we have that every non-faulty node q has raised $\{\text{A-ACCEPT}, G, m, \rho_0^q\}$ s.t. $rt(\rho_0^q) = r_0$ all within the round interval $[r' + \Delta_{4c}, r' + 2\Delta_{4c}]_{\Delta_{4c}}$.

From Corollary 8.14 we know that $r \in [r' + \Delta_{4c}, r' + 2\Delta_{4c}]_{\Delta_{4c}}$ as well. That is, for some $\Phi \in [0, \Delta_{4c}]_{\Delta_{4c}}$ the intervals $[r' + \Delta_{4c}, r' + 2\Delta_{4c}]_{\Delta_{4c}}$ and $[r - \Phi, r - \Phi + \Delta_{4c}]_{\Delta_{4c}}$ are the same, and we have the required. □

Validity

Lemma 8.13. Let G be a non-faulty node and let $r_0 \geq r_{clean}$. Has a non-faulty node p been running RUNNINGLOOP(G) in phase with (G, r_0) in the end of r, then:

(1) $r \geq r_0 + \Delta_{4c}$;

(2) and G has invoked StartAgreement in r_0.

Proof. (1) follows from Definition 8.1.

Assume to the contrary that G has not invoked StartAgreement in r_0 but some non-faulty node p is running RunningLoop(G) in phase with (G, r_0) in the end of $r \geq r_0 + \Delta_{4c}$.

From Corollary 8.3, and the *unforgeability* property of 4-cast, we have that no non-faulty node will raise $\{\overline{\text{4C-Accept}}, G, \langle \text{A-}init, * \rangle, 2\}$ in $r_0 + \Delta_{4c}$. Therefore, no node will use Line 8 to restart RunningLoop(G) to be in phase with (G, r_0) in $r \geq r_0 + \Delta_{4c}$.

Also, from Definition 8.1, no non-faulty node has been running RunningLoop(G) in phase with (G, r_0) before $r_0 + \Delta_{4c}$. Thus, from Claim 8.4 no non-faulty will 4-cast an A-*share* message in phase with (G, r_0) in r_0. Using Lemma 8.11, we have that the condition in Line 10 will not hold in phase with (G, r_0) for any non-faulty node in $r_0 + \Delta_{4c}$ and so no non-faulty will use Line 15 to restart RunningLoop(G) to be in phase with (G, r_0) in $r_0 + \Delta_{4c}$, either.

Combining with Corollary 8.8 we have that no non-faulty node will be running RunningLoop(G) in phase with (G, r_0) in the end of any round $\geq r_0 + \Delta_{4c}$, as required. □

Claim 8.11. Had a non-faulty G invoked StartAgreement with m in global round r_0, s.t. $r_0 \geq r_{clean} + (4f + 6)\Delta_{4c}$, then the condition in Line 5 will hold for every non-faulty node p in round $r_0 + \Delta_{4c}$ and the condition in Line 10 will hold for none.

Proof. Since $r \geq \iota_0$, G will execute Line 2 in round r_0 and 4-cast $\langle \text{A-}init, m \rangle$ in r.

Note that since $r_0 + \Delta_{4c} \geq r_{clean}$, from Corollary 8.3 and the *validity* property of 4-cast we have that every non-faulty node will raise $\{\overline{\text{4C-Accept}}, G, \langle \text{A-}init, m \rangle, 2\}$ in $r + \Delta_{4c}$.

Assume to the contrary that the condition in Line 5 did not hold for some non-faulty node p. Thus, p has either been running RunningLoop(G) in the beginning of $r_0 + \Delta_{4c}$ — in which case from Lemma 8.7 it has been running RunningLoop(G) in phase with $(G, \hat{r}_0)$ in some previous round $\geq r_0$; or p has ignored the A-*init* 4-cast received from G because it has executed Line 53 in some previous round $\geq r_0$ and then from Lemma 8.9 we have that p has been running RunningLoop(G) in phase with $(G, \hat{r}_0)$ in the end of some previous round $\geq r_0 - \Delta_{4c}$.

In any case, by applying Claim 8.8 we have that $\hat{r} \geq r_0 - \Delta_{4c} - (4f + 5)\Delta_{4c} \geq r_{clean}$.

However, from Lemma 8.13 this means that G has invoked StartAgreement in $\hat{r}$ in contradiction to the fact that G conforms with Definition 5.1 (as $\Delta^a_{wait} > (4f+6)\Delta_{4c}$).

As for the condition in Line 10. Note that we have shown that no non-faulty node has been running RunningLoop(G) in phase with $(G, *)$ in the end of any previous round $\geq r_0 - \Delta_{4c}$. From Claim 8.4 and Lemma 8.9, combined with Lemma 8.11 we have that the condition in Line 10 will not hold for any non-faulty node in $r_0 + \Delta_{4c}$. □

Corollary 8.15 (Validity)**.** Had a non-faulty G invoked StartAgreement with m in global round r_0, s.t. $r_0 \geq r_{clean} + (4f+6)\Delta_{4c}$, then in global round $r_0 + 2\Delta_{4c}$ every non-faulty node p would have raised $\{\text{A-Accept}, G, m, \rho^p_0\}$ s.t. $rt(\rho^p_0) = r_0$.

Proof. Note that from Claim 8.11 the condition in Line 5 will hold for every non-faulty node p in round $r_0 + \Delta_{4c}$. Therefore, from Corollary 8.1, in round $r + \Delta_{4c}$ every non-faulty node will restart RunningLoop(G) to be in phase with (G, r_0) in the end of $r_0 + \Delta_{4c}$ and will all set $v_p(G) = m$.

Also, no non-faulty node will restart RunningLoop(G) using Line 15 since the condition in Line 10 will not hold for any.

Using Claim 8.6 we have the required. □

Unforgeability

Corollary 8.16. Let p, G be non-faulty nodes, and let $r \geq r_{clean} + 2(4f+6)\Delta_{4c}$ be a global round. Had p raised $\{\text{A-Accept}, G, *, \rho^p_0\}$ in global round r, then G had invoked StartAgreement in $rt(\rho^p_0) = r - \Delta_{valid}$.

Proof. First, let $r_0 = rt(\rho^p_0)$.

From Lemma 8.9 we have that p must have been running RunningLoop(G) in phase with (G, r_0) in $r - \Delta_{4c}$, and from Claim 8.8 we have that $r_0 \geq r_{clean} + (4f+6)\Delta_{4c}$.

By applying Lemma 8.13 we have that G has invoked StartAgreement in r_0, and therefore, using Corollary 8.15 and Corollary 8.14 we have that $r = r_0 + \Delta_{valid}$. □

Properties and constants

Theorem 8.1 (Agreement)**.** Agreement satisfies the properties of the *Agreement* problem for $\Delta_{rmv} = (4f+4)\Delta_{4c} + 1$, $\Delta^a_{stb} = 2(6f+8)\Delta_{4c} + 1 + 2\Delta^{4c}_{stb} = O(f)$, $\Delta_{valid} = 2\Delta_{4c}$, $\Delta_{agr} = \Delta_{4c}$, $\Delta_{term} = (f+3)\Delta_{4c}$, $\Delta_{decay} = (4f+5)\Delta_{4c}$, $\Delta^a_{wait} = (4f+6)\Delta_{4c} + 1$.

Proof. The requirements regarding Δ_{rmv} in the previous claims are in Claim 8.8 and in Claim 8.9, and the maximal is $\Delta_{rmv} > (4f+4)\Delta_{4c}$ and so $\Delta_{rmv} = (4f+4)\Delta_{4c}+1$ is enough.

The only requirement regarding Δ^a_{wait} is in Claim 8.11 and is $\Delta^a_{wait} > (4f+6)\Delta_{4c}$ and so choosing $\Delta^a_{wait} = (4f+6)\Delta_{4c}+1$ is enough.

The requirements regarding Δ^a_{stb} are in the definitions of the claims regarding the actual properties, and the maximal is $\Delta^a_{stb} \geq \Delta_{rmv} + 2\Delta^{4c}_{stb} + 2(4f+6)\Delta_{4c}$. Putting in the requirement for Δ_{rmv} we have that $\Delta^a_{stb} = 2(6f+8)\Delta_{4c} + 1 + 2\Delta^{4c}_{stb}$ is enough. Note that assuming Δ_{4c} and Δ^{4c}_{stb} are constants we have that $\Delta^a_{stb} = O(f)$ as required.

The other constants are taken from the bodies of the appropriate claims. □

Corollary 8.17. Using the solution for 4-CAST presented previously in this thesis, we can choose $\Delta^a_{stb} = 6(6f+9)+1 = O(f)$.

Proof. Putting in $\Delta_{4c} = \Delta^{4c}_{stb} = 3$ in Theorem 8.1, we have $\Delta^a_{stb} = 2(6f+8)\Delta_{4c} + 1 + 2\Delta^{4c}_{stb} = 2(6f+8)3 + 7 = 6(6f+9)+1 = O(f)$ as required. □

Chapter 9

Appendix: Concept for the Semi-Synchronous Model

This appendix explains the general idea of how to translate node-invoked algorithms to the the semi-synchronous model.[1] The next paragraphs explain how to translate general algorithms using a translated 4-CAST primitive, and the last paragraph addresses the translation of 4-CAST itself.

Intuitively, what we propose is to generate an approximation to the global beat system around the initial invocation message that will be sent using 4-CAST.

The invocation message will be encapsulated in a 4-cast message, and so the difference between the invocation times the correct nodes will associate with it will be bounded. The time returned by this first 4-cast signal will be used as an approximation to the "first global beat", and each node will subsequently generate the next beats for itself. The time between beats will have to account for the bound on the difference between the invocation time approximations, and so will the width of the window in which nodes will look for messages assumed to be sent in the previous "round".

We expect this width to be around a small constant factor of the synchrony constant of the translated 4-CAST (which we expect to be a linear function of the network delay and clock drift bound with small constant factors).

As for how to translate 4-CAST itself, the idea is the same and a proposed algorithm is provided as Algorithm 9.1.[2] Note that this algorithm has been written intuitively and its correctness has not been proved or tested. The constants appearing probably

[1] The model with the constant bound on message traversal time.

[2] The constant d appearing denotes the bound on message delivery time, as defined in [15].

need to be tweaked, but we believe the concept to be correct.

Algorithm 9.1 4-CAST

```
 1: procedure START4-CAST(m)                                   ; ran by the sender G
 2:     send ⟨4C-init, m⟩ to all                               ; phase 1
 3: end procedure

 4: procedure 4C-JOININ(G)                                     ; ran by p continuously for every G ∈ P
 5:     if received ⟨4C-flood, G, m⟩ from ≥ t + 1 nodes in (ρ_p − 4d, ρ_p] then   ; phase 3
 6:         if received ⟨4C-flood, G, m⟩ from ≥ n − t nodes in (ρ_p − 3d, ρ_p − 1] then
 7:             schedule to send ⟨4C-support, G, m⟩ to all in ρ_p + 2d
 8:         end if
 9:         timeout_p(G) ← ρ_p + 4d
10:     end if
11:     if received ⟨4C-init, m⟩ from G and ρ_p > timeout_p(G) then   ; phase 2
12:         schedule to send ⟨4C-flood, G, m⟩ to all in ρ_p + d
13:         timeout_p(G) ← ρ_p + 4d
14:     end if
                                                               ; phase 4
        let maj_1 be the value received the most amongst 4C-support messages received in (ρ_p − 5d, ρ_p]
        let #maj_1 be the number of occurences of maj_1
        let maj_2 be the value received the most amongst 4C-support messages received in
                                                                   (ρ_p − 4d, ρ_p − 1]
        let #maj_2 be the number of occurences of maj_2
15:     if #maj_2 ≥ n − t then
16:         raise {4C-ACCEPT, G, m, 2}; timeout_p(G) ← −∞
17:     else if #maj_1 ≥ t + 1 then
18:         raise {4C-ACCEPT, G, m, 1}; timeout_p(G) ← −∞
19:     else if ρ_p = timeout_p(G) then
20:         raise {4C-ACCEPT, G, ⊥, 0}; timeout_p(G) ← −∞
21:     end if
22: end procedure

23: procedure 4C-CLEANUP                                       ; ran by p in the beginning of each round
24:     if timeout_p(G) > ρ_p + 4d then
25:         timeout_p(G) ← −∞
26:     end if
27: end procedure
```

Bibliography

[1] Ariel Daliot, Danny Dolev, and Hanna Parnas. Linear time byzantine self-stabilizing clock synchronization. In *Proceedings of 7th International Conference on Principles of Distributed Systems (OPODIS-2003), La*. Springer, 2003.

[2] Ariel Daliot and Danny Dolev. Self-stabilization of byzantine protocols. In *Proc. of the 7th Symposium on Self-Stabilizing Systems (SSS'05 Barcelona*, pages 48–67, 2005.

[3] Ezra N. Hoch. Self-stabilizing byzantine pulse and clock synchronization. Master's thesis, The Hebrew University of Jerusalem, Israel, 2007.

[4] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 148–161, New York, NY, USA, 1988. ACM.

[5] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Fast self-stabilizing byzantine tolerant digital clock synchronization. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, PODC '08, pages 385–394, New York, NY, USA, 2008. ACM.

[6] Cynthia Dwork and Larry Stockmeyer. Flipping persuasively in constant time. *SIAM J. Comput.*, 19:472–499, June 1990.

[7] Paul Feldman and Silvio Micali. An optimal probabilistic algorithm for synchronous byzantine agreement. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, pages 341–378, London, UK, 1989. Springer-Verlag.

[8] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27:228–234, April 1980.

[9] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

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

[11] Jonathan Katz and Chiu yuen Koo. On expected constant-round protocols for byzantine agreement. In *In Advances in Cryptology — Crypto '06*, pages 445–462. Springer-Verlag, 2006.

[12] Michael J. Fischer, Nancy A. Lynch, and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1981.

[13] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Simple gradecast based algorithms. *CoRR*, abs/1007.1049, 2010.

[14] S. Dolev. *Self-Stabilization*. MIT Press, March 2000.

[15] Ariel Daliot and Danny Dolev. Self-stabilizing byzantine agreement. *CoRR*, abs/0908.0160, 2009.

[16] Danny Dolev and Ezra N. Hoch. Byzantine self-stabilizing pulse in a bounded-delay model. In *Proceedings of the 9h international conference on Stabilization, safety, and security of distributed systems*, SSS'07, pages 234–252, Berlin, Heidelberg, 2007. Springer-Verlag.

[17] ITU-T Rec. X.667 — ISO/IEC 9834-8. Generation and registration of universally unique identifiers (uuids) and their use as asn.1 object identifier components. 2004.