I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1556

SIMPLE AND EFFICIENT ORACLE-BASED CONSENSUS PROTOCOLS FOR ASYNCHRONOUS BYZANTINE SYSTEMS

ROY FRIEDMAN , ACHOUR MOSTEFAOUI , MICHEL RAYNAL

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

Simple and Efficient Oracle-Based Consensus Protocols for Asynchronous Byzantine Systems

Roy Friedman , Achour Mostefaoui , Michel Raynal

Thème 1 — Réseaux et systèmes
Projet Adept

Publication interne n ° 1556 — Septembre 2003 — 26 pages

Abstract: This paper is on the Consensus problem in asynchronous distributed systems where (up to f) processes (among n) can exhibit a Byzantine behavior, i.e., can deviate arbitrarily from their specification. A way to solve the consensus problem in such a context consists of enriching the system with additional oracles that are powerful enough to cope with the uncertainty and unpredictability created by the combined effect of Byzantine behavior and asynchrony. Considering two types of such oracles, namely, an oracle that provides processes with random values, and a failure detector oracle, the paper presents two families of Byzantine asynchronous consensus protocols. Two of these protocols are particularly noteworthy: they allow the processes to decide in one communication step in favorable circumstances. The first is a randomized protocol that assumes $n > 5f$. The second one is a failure detector-based protocol that assumes $n > 6f$. These protocols are designed to be particularly *simple* and *efficient* in terms of communication steps, the number of messages they generate in each step, and the size of messages. So, although they are not optimal in the number of Byzantine processes that can be tolerated, they are particularly efficient when we consider the number of communication steps they require to decide, and the size of the messages they use. In that sense, they are practically appealing.

Key-words: Asynchronous distributed system, Byzantine process, Distributed algorithm, Fault tolerance, Random oracle, Randomized protocol, Unreliable failure detector.

(Résumé : tsvp)

email: {rfriedma,mostefaoui,raynal}@irisa.fr

Centre National de la Recherche Scientifique
(UPRESSA 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Protocoles de consensus simples et efficaces pour les systèmes asynchrones byzantins

Résumé : Cet article adresse le problème du consensus dans les systèmes répartis où un certain nombre de processus du système peuvent exhiber un comportement byzantin. Un comportement byzantin est un comportement qui dévie de la spécification du processus. Un moyen de résoudre le problème du consensus dans un tel contexte consiste à enrichir le système d'un oracle assez puissant pour permettre de limiter le "bruit" créé par l'effet combiné des processus byzantins et de l'asynchronie.

Nous avons étudié deux tels oracles, en l'occurrence, un générateur de nombres aléatoires et un détecteur de défaillances. Nous avons ainsi proposé deux familles de protocoles simples et efficaces en termes de nombre d'étapes de calcul, du nombre de messages échangés et de l'usage de signatures au niveau applicatif.

Mots clés : Algorithme probabiliste, algorithme réparti, consensus, détecteur de défaillances non fiable, processus byzantin, système asynchrone, tolérance aux fautes.

1 Introduction

Context and motivation A process has a *Byzantine* behavior when it behaves arbitrarily. This bad behavior can be intentional (malicious behavior) or more simply the result of a transient fault that altered the local state of the process, thereby modifying its behavior in an unpredictable way. We are interested here in solving agreement problems (more precisely, the *Consensus* problem) in asynchronous distributed systems prone to Byzantine process failures.

In the consensus problem, each process proposes a value, and the non-faulty processes have to decide (termination property) on the same output value (agreement property) that has to be related to the set of input values (validity property). This problem, whose statement is particularly simple, is fundamental in fault-tolerant asynchronous distributed computing as it abstracts several basic agreement problems. For example, Consensus can be used as a building block in implementing Atomic Broadcast[1], which is turn can be used as a building block for implementing a replicated state machine [15]. Unfortunately, the consensus problem has no deterministic solution in asynchronous distributed systems where even a single process can crash [7] – this is known as the FLP impossibility result. So, to solve consensus, asynchronous distributed systems have to be enriched with "additional power". Random oracles [1, 14] and unreliable failure detectors [4] are two such equipments that, when added to an asynchronous distributed system, make possible to solve consensus despite failures.

Contribution This paper presents two families of protocols for solving consensus in asynchronous systems prone to Byzantine process failures. One family is based on oracles providing random numbers, while the other family is based on failure detectors providing lists of processes suspected to be faulty. Like most consensus protocols, the ones that are presented here proceed in consecutive asynchronous rounds. Moreover, the design principles for both families of protocols are the same, namely, design simplicity on one hand and time and communication efficiency on the other hand. In the following, n denotes the total number of processes, while f denotes the maximum number of processes that can be faulty.

For the randomized protocols we propose, each round consists of one communication step, and in each such step, each process sends only one scalar to the other processes, in addition to the round number[2]. This means that a round costs $O(n^2)$ messages of size $O(\log_2 r)$ bits, where r is the round number. For didactic reasons, as well as simplicity and modularity of the presentation and proof, we develop our randomized protocol(s) as follows: We start by presenting a very simple randomized protocol that requires $n > 5f$. Next, we presents a slightly modified, very simple, protocol that allows terminating with one communication step in favorable circumstances (no process fails and all propose the same value), but requires $n > 7f$. This protocol is particularly interesting when the Byzantine failures are not intentional (malicious). In that case, failures are actually rare. Moreover, there are many situations where the processes start the consensus protocol already with the same value. In such circumstances, the proposed protocol is time optimal. Finally, these protocols are combined to get a randomized consensus protocol that has both of their advantages,

[1] Atomic Broadcast requires all non-faulty processes to deliver the same set of messages in the same order. Thus, it is both a *communication* problem (messages have to be delivered) and an agreement problem (there is a single delivery order). The agreement on the message delivery order is an instance of the consensus problem [4].

[2] Observe that, in asynchronous systems, messages have to carry the round number corresponding to the value they carry in order to prevent confusion. Differently, such a confusion is not possible in synchronous systems: as the message delays are bounded, messages from different rounds cannot coexist.

namely, $n > 5f$ and one step decision in good scenarios. So, this final protocol is constructed incrementally. Such a construction not only favors its understanding, but also allows for a modular proof.

Two failure detector-based consensus protocols are then proposed. The first is an adaptation to the asynchronous context of the simple and elegant Byzantine consensus protocol proposed by Berman and Garay [2] for synchronous systems. Berman-Garay's synchronous consensus protocol assumes $n > 4f$ and requires $2(f + 1)$ communication steps for the processes to decide. Our adaptation to asynchronous systems is subject to an additional price (namely, $n > 6f$ instead of $n > 4f$), but enjoys the property of allowing processes to decide in a single communication step in favorable circumstances (that is, when either all processes start with the same value and there are no failures, or at least $(n - f)$ correct processes start with the same value and the failure detector does not suspect any of them until each correct process received the first message from each of them).

Interestingly, in addition to being simple and efficient, (to our knowledge) this protocol is the first direct adaptation of a synchronous Byzantine consensus protocol to asynchronous systems. This adaptation shed additional light on the difference between synchronous systems and asynchronous systems equipped with failure detectors. In particular, it brings two immediate observations:

- First, in an asynchronous system, we cannot wait for more than $(n - f)$ processes without the risk of being blocked forever, which adds one f to the ratio between n and f. Moreover, as there can be up to f Byzantine processes, we in fact need to add $2f$ to the ratio between n and f, in order to compensate for the possible duplicity of Byzantine processes.
- Second, our protocol uses a muteness failure detector, i.e., a failure detector that can detect when processes stop sending protocol messages (to our knowledge, such failure detectors have been introduced in [6]). Yet, such a failure detector cannot detect if the messages contain legal values, whether the same message was sent to everyone, or if a message that was received was indeed supposed to be sent by the protocol. Synchronous systems, on the other hand, have a built-in muteness detection, as such systems inherently assume that protocol messages arrive in a timely manner.

The second failure detector-based protocol we propose improves the previous one in the sense that it assumes only $n > 4f$. But, each of its rounds is made up of four communication steps. Consequently, in favorable circumstances, it allows to decide only after four communication steps.

All the protocols we propose assume (1) a reliable communication network, and (2) the fact that a process cannot impersonate another process. In that sense, they do not rely on complicated mechanisms. However, they do not meet the $n > 3f$ lower bound [12]. This can be viewed as the price to be paid to favor simplicity and communication efficiency. As the cost of hardware goes down, it may be reasonable to prefer such solutions in some circumstances. For example, code simplicity implies shorter development time and lower chances for bugs. Fast termination means better service to clients, and reduced message and computation complexities mean improved throughput. For example, in order to tolerate one Byzantine failure, a protocol that assumes $n > 3f$ but whose complexity is $O(n^3)$ messages of size $O(n)$, used in a system of 4 processes, generates in each round 64 messages of size 4, and a total of 256 values are sent. For $n > 5f$, $n > 6f$, or even $n > 7f$, but with $O(n^2)$ messages per round we get 36, 49, and 64 messages of size one value, respectively. A similar calculation applies to the computational cost (CPU cycles) of running such consensus protocols.

Last but not least, being able to solve Byzantine consensus is important when one is interested in transient faults. As already noticed, such faults can entail an arbitrary behavior of some processes. In general, fault-tolerance can be achieved either by trying to mask faulty behavior, or by trying to detect and eliminate badly behaving processes. When consensus is used repeatedly, a process that is Byzantine (but not crashed) in one execution can be correct in a subsequent execution and vice versa. Thus, protocols that can mask Byzantine failures, rather than try to detect such processes and remove them, in the long run can be more robust, under the assumption that indeed no more than f processes suffer Byzantine behavior during each instance of the protocol.

Related work Byzantine failures have been introduced by Lamport, Shostack and Pease in the context of synchronous systems in [11, 12] where it is shown that $n > 3f$ is a necessary requirement. A synchronous consensus protocol that meets several consensus lower bounds has been proposed by Garay and Moses in [8], namely, their protocol assumes $n > 3f$, the processes terminate in $f+1$ rounds, and messages have a polynomial size. A synchronous protocol with constant message size has been proposed by Berman and Garay [2]. As already noticed, this protocol requires $n > 4f$ and processes decide in $2(f + 1)$ rounds.

The first randomized protocols to solve consensus in asynchronous Byzantine systems have been proposed by Ben-Or [1] and Rabin [14]. Ben-Or's protocol requires $n > 5f$, two communication steps per round, and $O(n^2)$ messages per round. Rabin's protocol requires $n > 10f$. None of them allows the processes to decide in one communication step in good circumstances.

Toueg presented a randomized asynchronous consensus protocol [16] that is optimal with respect to f (i.e., , it assumes $n > 3f$). This protocol requires three communication steps per round and $O(n^3)$ messages per round, each message being of size $O(n)$. This size is due to the fact that messages have to carry "proofs", those being made of "witness values", one per process.

Cachin, Kursawe, and Shoup present a randomized Byzantine consensus protocol [3] that assumes $n > 3f$, in which each round consists of two all-to-all communication steps and each message contains one or two RSA signatures. Their protocol is based on applying threshold cryptography (signatures) mechanisms at the protocol level. The fastest decision is after 3 communication steps (due to the fact that the protocol also uses one pre-phase communication step).

As already indicated, Doudou, Garbatino, Guerraoui, and Schiper have proposed the notion of *muteness* failure detectors [6]. They have also provided in the same paper an asynchronous consensus protocol based on such failure detectors. Doudou, Garbatino and Guerraoui [5] have later combined the use of such a failure detector with a certification mechanism and an echo broadcast primitive to obtain a Byzantine consensus protocol that is optimal with respect to f (i.e., $n > 3f$). Their protocol generates $O(n^2)$ messages of size at least $O(n)$ in each round. The protocol uses 4 communication steps in each failure free round. When the coordinator is suspected, they employ additional three communication steps in order to move to the next round.

Roadmap This paper is made up of five parts. Section 2 defines the asynchronous computation model and the consensus problems we are interested in. Then, Section 3 considers the case where the system is equipped with a random oracle and presents the randomized protocols. Section 4 considers the case where the system is equipped with a muteness failure detector and presents two failure detector-based protocols. Finally, Section 5 concludes the paper.

2 Computation Model and the Consensus Problem

2.1 Computation Model

Asynchronous system with Byzantine processes The system is made up of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, \ldots, p_n\}$, and each process proceeds at its own speed. Moreover, up to f processes can exhibit a *Byzantine* behavior[3], which means that such a process can behave in an arbitrary manner. This is the more severe process failure model: a Byzantine process can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc. A process that exhibit a Byzantine behavior is called *faulty*. Otherwise, it is *correct*.

Notation In the following, $\perp$ denotes a default value. Let V_i be an array of values; $\#_v(V_i)$ denotes the number of occurrences of the value v in the array V_i.

Asynchronous communication network The processes communicate by exchanging messages through a communication network, which is assumed to be reliable (no loss, creation, or corruption of messages). Moreover, there is no bound on message transfer delays.

We assume the existence of a one-to-all *broadcast* primitive. When a correct process issues *broadcast*(m), all processes receive the message m. On the other hand, when a faulty process p_i pretends to have issued *broadcast*(m), it is possible that it actually sent m_1 to some subset of processes, m_2 ($m_2 \neq m_1$) to another subset of processes, etc., and even no message at all to another subset of processes (this actually depends on the Byzantine behavior of p_i when it broadcasts m).

We also assume that a Byzantine process cannot impersonate the other processes. This can be easily realized with a network level authentication mechanism, e.g., a mechanism based on digital signatures (such as SSL, VPNs, etc.), or by the use of truly private communication links. This means that at the networking level, when a process p_i receives a message m from a sender that claims to be p_j, it can safely know whether the sender is indeed p_j or not. If p_j is not the sender, the network level of p_i can discard m.

Additional oracles Several approaches have been proposed to circumvent the FLP impossibility to solve consensus in asynchronous distributed systems prone to process failures [7]. The approach considered here consists in equipping the underlying system with additional oracles. Two such oracles are used in this paper, namely a random number generator (Section 3.1), and an unreliable failure detector (Section 4.1).

2.2 The Consensus Problem

The consensus problem has been informally stated in the Introduction. This paper considers *binary* consensus: every process p_i *proposes* a value $v_i \in \{0, 1\}$ and all correct processes have to *decide* on some value v, in relation to the set of proposed values. Let us observe that, in a Byzantine failure context, any Consensus definition cannot be too strong (e.g., it is not possible to force a faulty process to decide as a correct process, since a Byzantine process can decide whatever it wants; similarly it is not always possible to decide a proposed value since a faulty process can propose different values to distinct processes and consequently the notion of "proposed value" is

[3] When intentional, such behavior is called *malicious*.

not defined for Byzantine processes). Thus, in such a context, the *Consensus* problem is defined by the following three properties:

- Termination: Every correct process eventually decides.
- Agreement: No two correct processes decide different values.
- Validity: If all the correct processes propose the same value v, then no value different from v can be decided.

Probabilistic termination As indicated previously, one of our solutions to the Consensus problem in the presence of Byzantine processes is based on the use of a random oracle. So, the termination property can no longer be deterministic; it can only be ensured with some probability. Consequently, the previous termination property is weakened as follows [1, 14]:

- P-Termination: Every correct process decides with probability 1.

The *Randomized Consensus* problem is defined by the previous Validity, Agreement and P-Termination properties.

3 Randomized Byzantine Consensus

This section presents a random oracle-based protocol that solves the Randomized Consensus problem. This protocol, which is particularly simple, assumes $n > 5f$ and enjoys the property to allow the processes to terminate in a single communication step in favorable circumstances (i.e., when there are no failures and all the processes propose the same value), which is clearly time optimal. This protocol is presented in an incremental way. A randomized protocol that assumes $n > 5f$ is first presented. This protocol does not allow one step termination in good scenarios. Then, a variant of this protocol, that assumes $n > 7f$ and enjoys the one step termination property is presented. Finally, both versions are combined to get a protocol that has both of these properties (namely, $n > 5f$ and one step decision in good circumstances). This incremental presentation not only favors the understanding of the basic design principles, but also allows for a modular proof.

3.1 Random Oracle

This section considers the case where the system is equipped with a random oracle as defined by Rabin [14]. Such an oracle can be seen as a global entity that delivers a sequence of random bits $b_1, b_2, \ldots, b_r, \ldots$ to the processes (each bit b_r has the value 0 -or 1-, with probability 1/2).

More precisely, this oracle provides the processes with a primitive random () that returns a bit each time it is called by a process. In addition to being random, this bit has the following global property: the rth invocation of random () by any correct process p_i, returns it the bit b_r. This means the same random bit b_r is returned to each correct process as the result of its rth invocation of the random () primitive.

Remark. Such a random bit generator has been used in randomized consensus protocols (e.g., [14, 16]). An implementation of such a generator can be as follows: All the correct processes are equipped with (1) the same random number generator algorithm, and (2) the same seed [10]. It is easy to see that all processes get the same sequence of random bits. *End of remark.*

We assume in this section that the Byzantine processes cannot control the delivery order of messages by the network to correct processes. Appendix A motivates this assumption.

3.2 Underlying Principles

The first randomized protocol (i.e., the description of the behavior of a correct process) is described in Figure 1. It displays design principles that are used in its variants described in the next sections. A process p_i starts a consensus execution by invoking the function $\mathsf{Consensus}(v_i)$ where v_i is the value it proposes. It decides and terminates its execution when it executes the **return**(v) statement (line 108) that provides it with the decided value v.

The local variable est_i of a process p_i keeps its current estimate of the decision value (initially, $est_i = v_i$). The processes proceed by consecutive asynchronous rounds. Thus, the pair (r_i, est_i) of a correct process p_i describes its current state. The first part of the protocol is made up of the lines 101-104 that describes the initialization and the communication part of the current round. The second part is made up of the lines 105-110 that defines the management of the local estimate est_i and the decision rule. More precisely, we have:

- At each round r_i, a correct process p_i first gets the next random number and stores it in the local variable s_i (line 101), broadcasts the pair (r_i, est_i), and waits until it has got the current estimate of at least $n - f$ processes (let us observe that, as there are at most f faulty processes, no correct process can block forever at line 103). Then, p_i builds a vector V_i that represents its current view of the current estimate values.

- A simple test rules the second part: p_i checks if there is a value v that appears "often" in its current view of the estimate values (i.e., in the vector V_i). "Often" means here "at least $n - 2f$ times".

  - If there is such a value v, p_i adopts it as its new estimate (line 106). Moreover, p_i checks (line 107) if the value v is the same as the random value associated with this round (i.e., if $s_i = v$). If this condition is true, then p_i decides the value of its current estimate by executing **return**(est_i) (line 108). Otherwise, p_i proceeds to the next round.

    If p_i decides, it first broadcasts DEC(est_i) to provide the other processes with its last estimate (line 108). In that way, the other processes cannot be blocked in future rounds by the fact that p_i will no longer send messages. So, as soon as a process p_k receives DEC(est_i) from p_i at line 103 of round r, this DEC(est_i) message is used as the message p_i would have sent to p_k at any round $r' \geq r$ until p_k decides.
  - If there is no such value v, p_i adopts the random value s_i associated with the current round r_i as its new estimate (line 110) before proceeding to the next round.

3.3 Proof of the Protocol

Lemma 1 *Let us assume $n > 4f$, and consider the situation where, at the beginning of a round r, all the correct processes p_i have the same estimate value v (i.e., $est_i = v$). They will never change their estimates, thereafter.*

Proof Let r be a round such that all the correct processes p_i have $est_i = v$ at the beginning of r. As there are at most f faulty processes, each correct process p_i receives at least $n - f$ messages at line 103. It then follows that each correct process p_i receives at least $n - 2f$ messages that carry the value v, and the test of line 105 is satisfied for each correct p_i; consequently, each correct p_i sets est_i to v (let us notice that the value v is the single one to satisfy the test of line 105 as, due to the assumption $n > 4f$, v is then a majority value). Moreover, if p_i decides at line 108, it broadcasts

```
Function Consensus(v_i)

init: est_i ← v_i; r_i ← 0;

repeat forever
(101)  r_i ← r_i + 1; V_i ← [⊥, ..., ⊥]; s_i ← random();
(102)  broadcast VAL(r_i, est_i);
(103)  wait until ( VAL(r_i, −) or DEC(−) messages have been received
                                        from at least (n − f) distinct processes );
(104)  for each j: if (VAL(r_i, est_j) or DEC(est_j) received from p_j) then V_i[j] ← est_j endif;
(105)  if (∃v : v ≠ ⊥ : #_v(V_i) ≥ n − 2f)
(106)        then est_i ← v;
(107)             if (s_i = v)
(108)                then broadcast DEC(est_i); return (est_i) endif
(109)        else est_i ← s_i
(110)  endif
end repeat
```

Figure 1: Randomized Protocol Executed by p_i ($n > 5f$)

a DEC(v) message, thereby indicating to the other processes that its estimate will not change in the future. It follows that the estimates of the correct processes will never change in the all future rounds. $\square_{Lemma\ 1}$

Lemma 2 [Validity] *If all the correct processes propose the same value v, then no value $v' \neq v$ can be decided.*

Proof This lemma is an immediate consequence of Lemma 1, when we consider $r = 1$. As all estimates of correct processes remain equal to v, it follows from line 108 that no value $v' \neq v$ can be returned by a correct process. $\square_{Lemma\ 2}$

Lemma 3 *Let $n > 5f$. If there are two correct processes p_i and p_j such that the conditions $(\exists v : v \neq \bot : \#_v(V_i) \geq n - 2f$ and $(\exists w : w \neq \bot : \#_w(V_j) \geq n - 2f)$ hold at line 105 of a round r, then $v = w$.*

Proof Let V_i^r (resp., V_j^r) be the value of the local variable V_i (resp., V_j) after it has been updated at line 104 of round r. As (1) p_i is correct and $(\#_v(V_i) \geq n - 2f)$, (2) there are at most f Byzantine processes, we can conclude that any value $w \neq v$ is present in any vector V_j of a correct process p_j at most $3f$ times, i.e., $\forall w \neq v : \#_w(V_j) \leq 3f$ for any correct process p_j. Consequently, as $n > 5f$, a value $w \neq v$ can appear at most $n - 2f - 1$ times in V_j. In particular, since $(\exists w : w \neq \bot : \#_w(V_j) \geq n - 2f > n - 2f - 1)$, we conclude that $v = w$. $\square_{Lemma\ 3}$

Lemma 4 [Agreement] *Let $n > 5f$. No two correct processes decide different values.*

Proof Let r be the first round during which a correct process p_i decides. Let v be the value it decides. Moreover, in addition to the notation V_i^r previously introduced, let s_i^r be the value of the local variable s_i after it has been updated at line 101 of round r.

As p_i decides (at line 108), we conclude from the lines 105 and 107 that the condition $(\#_v(V_i^r) \geq n - 2f) \ \vee \ (s_i^r = v)$ is true. We consider two sub-cases according to the line (106 or 109) executed by a correct process p_j during r.

- p_j (correct) executes line 109. So, p_j sets est_j to s_j^r, which is equal to s_i^r (due to the property of the random oracle). Consequently, $est_j = v$.
- p_j (correct) executes line 106. In that case, p_j sets est_j to w where w is such that $(\#_w(V_j^r) \geq n - 2f)$. So, we have $(\#_v(V_i^r) \geq n - 2f) \wedge (\#_w(V_j^r) \geq n - 2f)$. From Lemma 3, we have $v = w$.

It follows that all the correct processes have the same estimate value v, and consequently, we conclude from Lemma 1 that no value v' different from v can be decided. $\square_{Lemma\ 4}$

Lemma 5 [P-Termination] *Each correct process decides with probability* 1.

Proof Let us first observe that no correct process remains blocked forever during a round r. This follows directly from the fact that, at r, a correct process p_i waits for $(n - f)$ VAL() or DEC() messages and there are at most f faulty processes.
Claim. With probability 1, there is a round r at the end of which all correct processes have the same estimate value. *End of the claim.*

Assuming the claim holds, it follows from Lemma 1 that all the correct processes keep their estimate value v and consequently the condition at line 105 (i.e., $\exists v : v \neq \perp : \#_v(V_i) \geq n - 2f$) is continuously true. By the assumption that the oracle is random, it follows that, with probability 1, there is eventually a round in which the oracle outputs v. Then, the condition of line 107 evaluates to true, and all correct processes decide.

Proof of the claim. We need to prove that, with probability 1, there is a round at the end of which all correct processes have the same estimate value. Let us consider a round r.

- let us first observe that if all the correct processes execute line 109 then, at the end of r, all correct processes adopt the same value (namely, the value ouptut by the random oracle). The claim directly follows.
- If all the correct processes execute line 106, then due to line 105 and Lemma 3, they adopt the same v as their estimate, and the claim follows.
- The third case is when some correct processes execute line 106 and adopt the same value v (by Lemma 3), while others execute line 109 and adopt the same value s.

  Let p be the smallest probability that, during a round (r) the value (v^r) adopted in line 106 is the same as the value (s^r) output by the oracle and adopted at line 109. Due to the assumption that Byzantine processes do not control the delivery of messages to correct processes, $p > 0$. Let $P(r)$ be following probability: $P(r) = \text{Probability}[\exists r' : r' \leq r : v^{r'} = s^{r'}]$. We have $P(r) = p + (1 - p)p + \cdots + (1 - p)^{r-1}p$. So, $P(r) = 1 - (1 - p)^r$. As $\lim_{r \to +\infty} P(r) = 1$, the claim follows.

End of the proof of the claim. $\square_{Lemma\ 5}$

Theorem 1 *Let $n > 5f$. The protocol described in Figure 1 solves the Randomized Consensus problem.*

Proof The proof follows from the Lemmas 2, 4 and 5. $\square_{Theorem\ 1}$

Theorem 2 *The cost of a round of the protocol is one communication step, and $O(n^2)$ messages. The size of a message sent at round r is $O(\log_2 r)$ bits.*

Proof The theorem follows directly from line 102: at each round, a correct process broadcasts a message. Moreover, a VAL() message carries a bit and a round number. $\square_{Theorem\ 2}$

3.4 One Step Decision in Favorable Circumstances

As noticed in the Introduction, although they can occur, (when not malicious) Byzantine failures are rare. So, it is interesting to have protocols that termiante very quickly in favorable circumstances. Those are defined as follows: no process fails and all processes propose the same value.

In a very interesting way, a slight modification to the previous protocol provides us with a simple variant that allows the processes to decide in a single communication step in favorable circumstances. This variant is described in Figure 2. Basically, this variant weakens the two conditions that allows a process to decide (to facilitate the presentation, those condition weakening have been placed into boxes) and strengthens the context in which it can be used (an additional price has to be paided, namely, $n > 7f$ instead of $n > 5f$; this context strengthening is only needed to ensure the Termination property).

More precisely, a process p_i must not wait for more than $(n - f)$ values (to prevent deadlock at line 203). Yet, we would like to allow p_i to decide in a single step, if it has seen the same value from "enough" processes. Hence the new deciding test includes $\#_v(V_i) \geq (n - f)$ at line 207. But now, when a correct process p_i sees $(n - f)$ times the same value v, another correct process p_j might see this value v only $(n - 3f)$ times: among the $(n - f)$ entries equal to v in V_i (assume without loss of generality that the other entries of V_i are equal to $\perp$), p_j can "see" f of them equal to $\perp$ (due to asynchrony and line 203), and f other entries containing values different from v (due to the possibility of up to f Byzantine processes). That is, if a process p_i decides v because $\#_v(V_i) \geq (n - f)$, then in order to ensure agreement, we would like other correct processes to also adopt v. For this, the minimal requirement is to adopt v if $\#_v(V_j) \geq (n-3f)$, hence the weakening of the test of line 205.

```
Function Consensus(v_i)

init: est_i ← v_i; r_i ← 0;

repeat forever
(201)  r_i ← r_i + 1; V_i ← [⊥, ... , ⊥]; s_i ← random();
(202)  broadcast VAL(r_i, est_i);
(203)  wait until ( VAL(r_i, −) or DEC(−) messages have been received
                                  from at least (n − f) distinct processes );
(204)  for each j: if (VAL(r_i, est_j) or DEC(est_j) received from p_j) then V_i[j] ← est_j endif;
(205)  if (∃v : v ≠ ⊥ : #_v(V_i) ≥ [n − 3f] ) % Weakening of the first condition
(206)        then est_i ← v;
(207)             if [(#_v(V_i) ≥ n − f) ∨] (s_i = v) % Weakenning of the second condition
(208)               then broadcast DEC(est_i); return (est_i) endif
(209)        else est_i ← s_i
(210)  endif
end repeat
```

Figure 2: A Fast Terminating Protocol ($n > 7f$)

The correctness proof of this protocol has exactly the same structure as the protocol described in Figure 1. Basically, the Lemmas 1, 2, and 5 are verbatim exactly the same (with only replacing $(n-2f)$ with $(n-3f)$ in the text of Lemma 5). Albeit the proofs of the other lemmas are very close, we restate them for consistency.

Lemma 6 *Let $n > 7f$. If there are two correct processes p_i and p_j such that the conditions $(\exists v : v \neq \bot : \ \#_v(V_i) \geq n-3f$ and $(\exists w : w \neq \bot : \ \#_w(V_j) \geq n-3f)$ hold at line 205 of a round r, then $v = w$.*

Proof Let V_i^r (resp., V_j^r) be the value of the local variable V_i (resp., V_j) after it has been updated at line 204 of round r. As (1) p_i is correct and $(\#_v(V_i) \geq n-3f)$, (2) there are at most f Byzantine processes, we can conclude that any value $w \neq v$ can be present in the vector V_j of any correct process p_j at most $4f$ times, i.e., $\forall w \neq v : \#_w(V_j) \leq 4f$ for any correct p_j. Consequently, as $n > 7f$, a value $w \neq v$ can appear at most $n-3f-1$ times in V_j. In particular, since $(\exists w : w \neq \bot : \ \#_w(V_j) \geq n-3f > n-3f-1)$, we conclude that $v = w$. $\square_{Lemma\ 6}$

Lemma 7 [Agreement] *Let $n > 7f$. No two correct processes decide different values.*

Proof Let r be the first round during which a correct process p_i decides. Let v be the value it decides. (The notation V_i^r and s_i^r are the same as in Lemma 4.) As p_i decides (at line 208), we conclude from line 207 that the condition $(\#_v(V_i^r) \geq n-f) \vee (s_i^r = v)$ is true. There are two cases.

- $(\#_v(V_i^r) \geq n-f)$ is satisfied. In this case, it follows from this condition and the fact that there are at most f faulty processes that, for any correct process p_j, we have $(\#_v(V_j^r) \geq n-3f)$. So, when p_j tests the condition at line 205 of r, this condition is satisfied and consequently p_j sets est_j to v. Note that, as $n > 7f$, a value appears at least $n-3f$ times in V_j^r is a majority value in V_j^r. So, no other value v' can satisfy the test.
- $(\#_v(V_i^r) < n-f) \wedge (s_i^r = v)$ is satisfied. Note that as p_i decides v, we have $(\#_v(V_i^r) \geq n-3f)$ from line 205. We consider two sub-cases according to the line (206 or 209) executed by a correct process p_j during r.
  - p_j (correct) executes line 209. So, p_j sets est_j to s_j^r, which is equal to s_i^r (due to the property of the random oracle). Consequently, $est_j = v$.
  - p_j (correct) executes line 206. In that case, p_j sets est_j to w where w is such that $(\#_w(V_j^r) \geq n-3f)$. So, we have $(\#_v(V_i^r) \geq n-3f) \ \wedge \ (\#_w(V_j^r) \geq n-3f)$. From Lemma 6, we have $v = w$.

It follows that all the correct processes have the same estimate value v, and consequently, we conclude from Lemma 1 that no value v' different from v can be decided. $\square_{Lemma\ 7}$

Theorem 3 *Let $n > 7f$. The protocol described in Figure 2 solves the randomized Consensus problem. The cost of a round of the protocol is one communication step, and $O(n^2)$ messages. The size of a message sent at round r is $O(\log_2 r)$ bits. When, there is no failure and all processes propose the same value, the processes decide in one communication step.*

Proof The proof follows from Lemmas 2, 5 and 7. The proof of the cost is the same as in Theorem 2. The one communication step in favorable circumstances follows directly from the protocol. $\square_{Theorem\ 3}$

3.5 Early Decision when $n > 5f$

Interestingly, it is possible to combine the two previous randomized protocols to get a protocol that benefits from the best of each of them, i.e., terminates in a single communication step when there are no failures and all processes propose the same value, *and* requires only $n > 5f$. This combination, described in Figure 3, is very simple. It is as follows. Each correct correct process p_i executes:

- During the first round, the early deciding protocol (Figure 2) without the test $s_i = v$ at line 207,
- During the next rounds, the first protocol (Figure 1).

It is easy to see that the processes terminate in a single communication step when there is no failure and all processes propose the same value. Moreover, if no correct process terminates during the first round, the protocol behavior is the same as the one of Figure 1, and the correctness proof follows from Theorem 1. So, we only need to show that if some correct process decides v during the first round, no other value can be decided.

```
Function Consensus(v_i)

init: est_i ← v_i; r_i ← 0;

repeat forever
(301)  r_i ← r_i + 1; V_i ← [⊥, ... , ⊥]; s_i ← random();
(302)  broadcast VAL(r_i, est_i);
(303)  wait until ( VAL(r_i, −) or DEC(−) messages have been received
                                    from at least (n − f) distinct processes );
(304)  for each j: if (VAL(r_i, est_j) or DEC(est_j) received from p_j) then V_i[j] ← est_j endif;
(305)  case r_i = 1 : if (∃v : v ≠ ⊥ : #_v(V_i) ≥ n − 3f)
(306)                 then est_i ← v;
(307)                      if (#_v(V_i) ≥ n − f)
(308)                        then broadcast DEC(est_i); return (est_i) endif
(309)                 else est_i ← s_i
(310)            endif
(311)       r_i > 1 : if (∃v : v ≠ ⊥ : #_v(V_i) ≥ n − 2f)
(312)                 then est_i ← v;
(313)                      if (s_i = v)
(314)                        then broadcast DEC(est_i); return (est_i) endif
(315)                 else est_i ← s_i
(316)            endif
(317)  endcase
end repeat
```

Figure 3: Fast Termination with $n > 5f$

Lemma 8 *Let us consider the protocol described in Figure 3. Let $n > 5f$. If a correct process decides v during the first round, no value $w \neq v$ can be decided by another process in a later round.*

Proof Let p_i be a correct process that decides v during the first round. It decides at line 308, and consequently, we have $(\#_v(V_i) \geq n - f)$ (line 307). Let p_j be a correct process that does not decide during the first round. *Claim:* At the end of the first round, $est_j = v$. *End of the claim.* Once we

establish this claim, the theorem follows from Lemma 4.

Proof of the claim. As (1) $(\#_v(V_i) \geq n - f)$, (2) there at most f Byzantine processes, (3) p_j waits for messages from $(n - f)$ processes, we have (A): $(\#_v(V_j) \geq n - 3f)$. Hence, the test of line 305 is satisfied for p_j.

Moreover, as $(\#_v(V_i) \geq n - f)$, we can conclude that at least $(n - 2f)$ correct processes have proposed v. This means that (B): at most $2f$ values are different from v and $\perp$ in any vector V_j.

It follows from (A), (B) and the asumption $n > 5f$ that there is a single value v such that $(\#_v(V_j) \geq n - 3f)$. Consequently, p_j sets est_j to v at line 306. *End of the proof of the claim.* $\square_{Lemma\ 8}$

4 Failure Detector-Based Byzantine Consensus

This section presents two Byzantine consensus protocols based on a failure detector oracle. In both protocols, the processes proceed by consecutive asynchronous rounds. The first protocol has the additional property that it allows the processes to decide in one communication step in favorable circumstances. It requires $n > 6f$. The second protocol requires only $n > 4f$, but each round is made up of four communication steps (and the earliest decision is after 4 communication steps).

4.1 A Class of Unreliable Failure Detectors

This section assumes that the underlying system is equipped with a failure detector that is able to detect mute processes, the definition of which is based on the notion of protocol message[4]. A *protocol message* is a message that could have been sent during an execution of the protocol. For example, when we consider the previous randomized protocols, VAL(r, v) messages (for any value of r) and DEC(v) messages are the only protocol messages that can be generated during an execution. In that case, VAL(r) (or DEC) is considered as the *header* of the message, while v is the *data part* of the message. The syntax of a protocol message corresponds to the syntax of a message generated by the protocol. If that message has been generated by a Byzantine process it can carry a wrong data (e.g., when considering the previous protocols, the data v in VAL(r, v) or DEC(v) can be wrong).

A process is *mute* (with respect to a protocol) if after some time it does not send (some or all) protocol messages (with headers) it was assumed to send. As an example, a process that, during a round r, never sends a message it is assumed to send during that round, is mute. We emphasize that a Byzantine process is not mute if, during each round, it sends all the messages with headers that would have been been sent if it was correct. Of course, as it is Byzantine, the data part of those messages can be wrong. Let us observe that the set of mute processes also includes all crashed processes. To cope with mute processes, we consider the class of failure detectors, denoted $\diamond\mathcal{P}_{mute}$, that includes all the failure detectors that satisfy the following properties:

- Muteness Strong Completeness: Eventually, every mute process is permanently suspected by every correct process.
- Eventual Strong Accuracy: There is a time after which no correct process is suspected.

Let us notice that failure detectors of the class $\diamond\mathcal{P}_{mute}$ can be implemented in partially synchronous systems [6].

[4]To our knowledge, these notions (*protocol message* and *mute process*) have been introduced in [6]. Our definition of *mute process* -see below- is stated slightly different than the one described in [6].

4.2 An Early Deciding Protocol: Underlying Principles

The proposed $\diamond\mathcal{P}_{mute}$-based Byzantine consensus protocol is an adaptation to Byzantine asynchronous systems of the simple and elegant consensus protocol proposed by Berman and Garay for synchronous systems [2]. As noticed in the Introduction, this adaptation keeps the simplicity of the original synchronous protocol, but requires a stronger assumption (namely $n > 6f$ instead of $n > 4f$) to cope with asynchrony. Moreover, as previously noticed, it has the nice property of allowing the processes to decide in a single communication step when no process is faulty and all processes propose the same value.

As in the previous randomized protocol, the processes proceed in consecutive asynchronous rounds. Each round is made up of two phases. The first phase of each round is a global exchange, while the second phase is based on the rotating coordinator paradigm.

In addition to Termination and Agreement, a Byzantine protocol must ensures Validity, namely, the decided value has to be v when all correct processes propose v. To attain this goal, the protocol is based on the following principle: (1) If the occurrence number of the most current value bypasses some threshold, that value will be decided. (2) Otherwise, the coordinator paradigm is used to eventually force a value to be adopted by enough processes so that the previous property becomes satisfied. This principle is implemented as follows:

- During the first phase of a round r, the processes exchange their estimates est_i of the decision value (line 402), and build their local view of current estimates (line 405). Let us notice that the DEC(-) messages play the same role as in the previous protocol. Finally, each process p_i determines, and stores in $dominating_i$, the majority estimate if any (otherwise, $dominating_i$ is set to est_i).

- The aim of the second phase is to allow the processes to decide, or if they cannot, to adopt the same estimate value for the next round.

  To that end, p_i adopts, as its new estimate, the value $dominating_i$ if it is "present enough" in V_i (line 408). "Present enough" means that it appears at least $(n - 2f - \#_\perp(V_i))$ times. Moreover, if the value $dominating_i$ is "very present" in V_i, p_i decides it (line 409). "Very present" means that it appears at least $n - f$ times. The proof gives insight as to why these threshold values, namely, $(n - 2f - \#_\perp(V_i))$ and $(n - f)$, have been chosen.

  Otherwise (the value $dominating_i$ is not present enough), the round-based coordinator paradigm is used to force the processes p_i to adopt the same value. More precisely, during each round, the corresponding round coordinator broadcasts its dominating value v (line 406) to try to impose it to the other processes. Each process p_i that receives it, stores it in $coord\_val_i$ (line 412). If it does not receive it or suspects the coordinator, p_i uses its current dominating value as a default value (line 413). Then, p_i adopts the resulting $coord\_val_i$ as its new estimate (line 415).

4.3 Proof of the Protocol

The proof has the same structure as the proof in Section 3.3. The main differences are in the proof of the Agreement and Termination properties.

Lemma 9 *Let us assume $n > 4f$, and consider the situation where, at the beginning of a round r, all the correct processes p_i have the same estimate value v (i.e., $est_i = v$). They will never change their estimates, thereafter.*

```
Function Consensus(v_i)

init: est_i ← v_i; r_i ← 0;

repeat forever
(401)  r_i ← r_i + 1; V_i ← [⊥, ..., ⊥]; c ← ((r_i − 1) mod n) + 1;
       ——————————————— Step 1 of round r ———————————————
(402)  broadcast VAL(r_i, est_i);
(403)  wait until ( VAL(r_i, −) or DEC(−) messages have been received from all non-suspected
                    processes and from at least (n − f) distinct processes );
(404)  for each j: if (VAL(r_i, est_j) or DEC(est_j) received from p_j) then V_i[j] ← est_j endif;
(405)  if (∃v ≠ ⊥ : #_v(V_i) > n/2) then dominating_i ← v else dominating_i ← est_i endif;
       ——————————————— Step 2 of round r ———————————————
(406)  if i = c_r then broadcast COORD(r_i, dominating_i) endif;
(407)  if (#_{dominating_i}(V_i) ≥ n − 2f − #_⊥(V_i))
(408)        then est_i ← dominating_i;
(409)             if (#_{dominating_i}(V_i) ≥ n − f) then broadcast DEC(est_i); return (est_i) endif
(410)        else wait until (COORD(r_i, −) or DEC(−) received from p_c or p_c is suspected);
(411)             if (COORD(r_i, x) or DEC(x) received from p_c)
(412)                  then coord_val_i ← x
(413)                  else coord_val_i ← dominating_i
(414)             endif;
(415)             est_i ← coord_val_i
(416)  endif
end repeat
```

Figure 4: $\diamond P_{mute}$-Based Protocol Executed by p_i ($n > 6f$)

Proof Note that in every round, each correct process collects at least $(n - f)$ estimates. Since at the beginning of round r, all correct processes have v as their initial estimate and as there are at most f Byzantine processes, then every correct process p_i will collect at least $n - 2f$ estimates equal to its own estimate v. As $n > 4f$, it follows that v is a majority value in V_i and therefore $dominating_i$ is set to v (line 405). Hence, est_i is set to $dominating_i = v$ (line 408). $\square_{Lemma\ 9}$

Lemma 10 [Validity] *If all the correct processes propose the same value v, then no value $v' \neq v$ can be decided.*

Proof The proof of this lemma is left to the reader: it is verbatim the proof of Lemma 2 (with the exception of the line number, and replacing Lemma 1 with Lemma 9). $\square_{Lemma\ 10}$

Lemma 11 [Agreement] *Let $n > 6f$. No two correct processes decide different values.*

Proof Let r be the first round during which a correct process p_i decides, and let v be the value it decides. Due to the lines 405 and 409, it follows that $dominating_i = v$ and $\#_v(V_i) \geq n - f$. Due to the fact that at most f processes are Byzantine, it follows that, in the worst case, p_j sees the same values as p_i except for $\#_\perp(V_j)$ entries that are equal to $\perp$ in V_j (those being equal to v in V_i), and at most f other entries (those possibly corresponding to Byzantine processes that sent v to p_i and $v' \neq v$ to p_j). It follows that $\#_v(V_j) \geq n - f - (f + \#_\perp(V_j))$, i.e., $\#_v(V_j) \geq n - 2f - \#_\perp(V_j)$ for any correct process p_j. As $\#_\perp(V_j) \leq f$, we get $\#_v(V_j) \geq n - 3f$ and, as $n > 6f$, it follows that v is a majority value in V_j. Hence, $dominating_j = v$ (line 405).

Moreover, as $\#_v(V_j) \geq n - 2f - \#_\perp(V_j)$, the test at line 407 is satisfied for any correct process p_j and, accordingly, any correct p_j sets est_j to v at line 408. If p_j decides at line 409, it decides v. If p_j proceeds to the next round, due to Lemma 9, no value $v' \neq v$ can be decided. $\square_{Lemma\ 11}$

Lemma 12 *No correct process can block forever in a round.*

Proof The lemma follows immediately from the following observations. At each round r:
- As there are as most f faulty processes, no correct process can block forever at line 403,
- As the failure detector satisfies the Muteness Strong Completeness property, no correct process can block forever at line 410. $\square_{Lemma\ 12}$

Lemma 13 [Termination] *Let $n > 6f$. Each correct process eventually decides.*

Proof Let t be the time after which the failure detector is accurate, i.e., no correct process is suspected (due to the Eventual Strong Accuracy of the failure detector, such a time t does exist). Let r be the first round that starts after t and is coordinated by a correct process p_c. Let us observe that, due to Lemma 12 and the use of DEC() messages (if any), any correct process p_i that has not yet decided starts round r. During r, let $dominating_c = v$.
Claim. At the end of r (where $dominating_c = v$), all correct processes p_i have $est_i = v$. *End of the claim.*

Due to the claim, it follows that all the correct processes (that have not yet decided) start the round $r + 1$ with the same estimate value v. Moreover, due to (1) the fact that there are at least $(n - f)$ correct processes, (2) the fact that the failure detector is accurate (i.e., no correct process is suspected), (3) the DEC () messages sent by the processes that have already decided (if any), and (4) the waiting statement of line 403 (messages are received from all correct processes), it follows that all the correct processes p_i are such that $\#_v(V_i) \geq n - f$, and v is the only such value (because $n - f > f$). So, for any correct p_i, we have $dominating_i = v$ at line 405. Consequently, both tests of lines 407 and 409 are satisfied, and each correct process p_i decides accordingly by the end of $r+1$.

Proof of the claim. Let us first observe that if each correct process executes line 415, it adopts v as its new estimate and the claim trivially follows.

Let us consider the case where a process p_i executes line 408, namely, $est_i \leftarrow dominating_i$. Let $dominating_i = w$. We have to show that $v = w$. As p_i executes line 408, the test of line 407 is satisfied and we have $\#_w(V_i) \geq n - 2f - \#_\perp(V_i)$. Moreover, as (1) p_i is correct, (2) there are at most f faulty processes, (3) we are after the time t (and consequently, each correct process receives a message from each correct process), we can conclude that the entries k such that $V_i[k] = \perp$ correspond to faulty processes. Consequently, for any correct process p_j, we have $\#_w(V_j) \geq n - 2f - \#_\perp(V_i) - (f - \#_\perp(V_i))$, i.e., $\#_w(V_j) \geq n - 3f$. So, when we consider the coordinator p_c, we get $\#_w(V_c) \geq n - 3f$. As $n > 6f$, we have $\#_w(V_c) \geq n - 3f > n/2$, and so w is a majority value in the vector V_c. It then follows from line 405, that $dominating_c = w$. Hence $w = v$. It follows that all correct processes p_i have $est_i = v$ at the end of r. *End of proof of the claim.* $\square_{Lemma\ 11}$

Theorem 4 *Let $n > 6f$. The protocol described in Figure 4 solves the Consensus problem.*

Proof The proof follows from the Lemmas 10, 11 and 13. $\square_{Theorem\ 4}$

Theorem 5 *The cost of a round of the protocol is at most two communication steps, and $O(n^2)$ messages. The size of a message sent at round r is $O(\log_2 r)$ bits.*

Proof The theorem follows directly from the protocol code. Each round is made up of two phases: the first phase consists of n broadcasts, while the second phase has a single broadcast. Each message carries a proposed value. Moreover, VAL() messages carry a round number. $\square_{Theorem\ 5}$

Theorem 6 *If (1) no process is faulty and all processes propose the same value v, or (2) at least $(n-f)$ correct processes start the execution with the same value v, and the failure detector does not suspect any of them until all correct processes received the first message from each of them, then each correct process decides in one communication step.*

Proof The proof of this theorem is left to the reader. (It is nearly an immediate consequence of the protocol code.) $\square_{Theorem\ 6}$

Interestingly, the requirement concerning the behavior of the failure detector, stated in item (2) of the previous theorem, is likely to be met by most reasonable implementations of failure detectors when either the failure detector starts executing at the same time as the protocol, or in periods when the network behaves well. Alternatively, the condition stated in item (1) ensures termination even if the failure detector is misbehaving (e.g., when the network is unstable). In that sense, this theorem states a tradeoff relating the number of faulty processes and the failure detector behavior, to terminate in a single communication step when correct processes propose the same value v.

4.4 Favoring Fault-Tolerance: Underlying Principles

This section presents a second consensus protocol based on $\diamond P_{mute}$. When compared to the protocol described in Figure 4, this protocol improves on the bound f, namely it requires $n > 4f$ instead of $n > 6f$. This improvement is obtained at the price of longer rounds. More explicitly, (as in the previous protocols) the processes proceed in consecutive asynchronous rounds, and each round is now made up of four communications steps. It follows that, in favorable circumstances, this protocol allows the processes to decide in 4 communication steps.

Both the previous protocol (Figure 4) and this one use the rotating coordinator paradigm. But, they use it in a very different way. In the previous protocol, the coordinator of the current round is used only in the "extreme case", where no value is "present enough" in the vector V_i (see line 407 and lines 410-415 in Figure 4); otherwise, it is not used (see line 408). This means that there are executions where the round coordinators are not used at all. Very differently, in the protocol presented in Figure 5, during each round, each process waits for a message from the round coordinator, and the estimate value adopted by a process during a round depends on the content of the message it receives from the round coordinator. Hence, the core of this protocol consists in ensuring that, when a process adopts the current coordinator estimate, that value is a plausible decision value despite the fact that the coordinator can be Byzantine. So, although both the protocols are coordinator-based, they rely on very distinct design principles.

As already indicated, each round of the protocol is made up of 4 communication steps. They work as follows:

- The aim of the first step (lines 502-504) is to provide each process p_i with a local view V_i of the current estimates.

- The second step (lines 505-514) is where the round coordinator is used. Each process p_i waits for the view of the coordinator (line 507), and tries to adopt (in a local variable x_i) a value suggested by the round coordinator (line 509). This value is the one that appears most frequently in the view V_i^c it has received from the coordinator. Ties are broken, e.g., by choosing 1 when 1 and 0 appear the same number of times. The aim is for all the correct processes to adopt the same value, despite the fact that the current coordinator can be Byzantine and have sent different vectors to different processes. A first check to attain this goal is done at line 508: p_i first tests if its local view V_i and the view V_i^c it received from the coordinator share at least $(n-f)$ non-$\perp$ entries (i.e., $(n-f)$ entries k such that $V_i[k] = V_i^c[k] \neq \perp$). If it is the case, p_i trusts p_c and adopts the most frequent value in V_i^c. Otherwise (line 510) p_i adopts $\perp$.

  If p_i received DEC(w) from p_c, it adopts w only if w is the current estimate of at least one correct process, i.e., if $\#_w(V_i) \geq (f+1)$. Otherwise, it does not trust p_c and adopts $\perp$ (line 512).

- The third step (lines 515-518) is a "cleaning phase". More precisely, it is possible that two or more correct processes p_i and p_j trust the current coordinator but adopt different values (i.e., $x_i = v \neq \perp$ and $x_j = v' \neq \perp$, with $v \neq v'$). This can result from (1) asynchrony (a process cannot wait for more than $(n-f)$ messages without risking to be blocked forever at line 503 when it builds it local view V_i), and (2) the fact that the current coordinator p_c is Byzantine and sends different V^c vectors to distinct correct processes.

  So, the aim of the third step is to end up in a situation in which there is a single value v such that every correct process either supports v or $\perp$. This is obtained by a global exchange, after which it is ensured that all correct processes that do not adopt $\perp$ adopt the same v. This is because a process adopts a value v only if v appeared in at least $(n-2f)$ ECHO(r_i, v) messages sent by correct processes, i.e., as $n > 4f$, by at least a majority that includes only correct processes. As there is a single majority, there is a single such v. It then follows that, at the end of the third phase, the set of the $coord\_val_i$ values of the correct processes is such that

  $$(coord\_val_i \neq \perp \wedge coord\_val_j \neq \perp) \;\Rightarrow\; (coord\_val_i = coord\_val_j).$$

- The last step of a round (lines 519-525) is a phase during which the correct processes exchange the value (v or $\perp$) they have just adopted. If the value v has been received "very often" (i.e., at least $(n-f)$ times) then p_i decides this value (line 522). If the value v has been received "often enough" (i.e., at least $(n-3f)$ times) then p_i adopts it as its new current estimate before entering the next round (line 523). Otherwise, p_i keeps its previous estimate. As there are at most f Byzantine processes, they cannot corrupt the previous updates. That is, even if they send the same $v' \neq v$, such a v' will appear in at most f CONFIRM/DEC() messages, and therefore $\#_{v'}(W_i) \leq f$.

  It follows that the combined use of the thresholds $(n-f)$ (line 522) and $(n-3f)$ (line 523) ensures that if a correct process decides v during a round r, then no other value can be decided by a correct process at a round $r' \geq r$.

4.5 Proof of the Protocol

The structure of this proof resembles the previous ones.

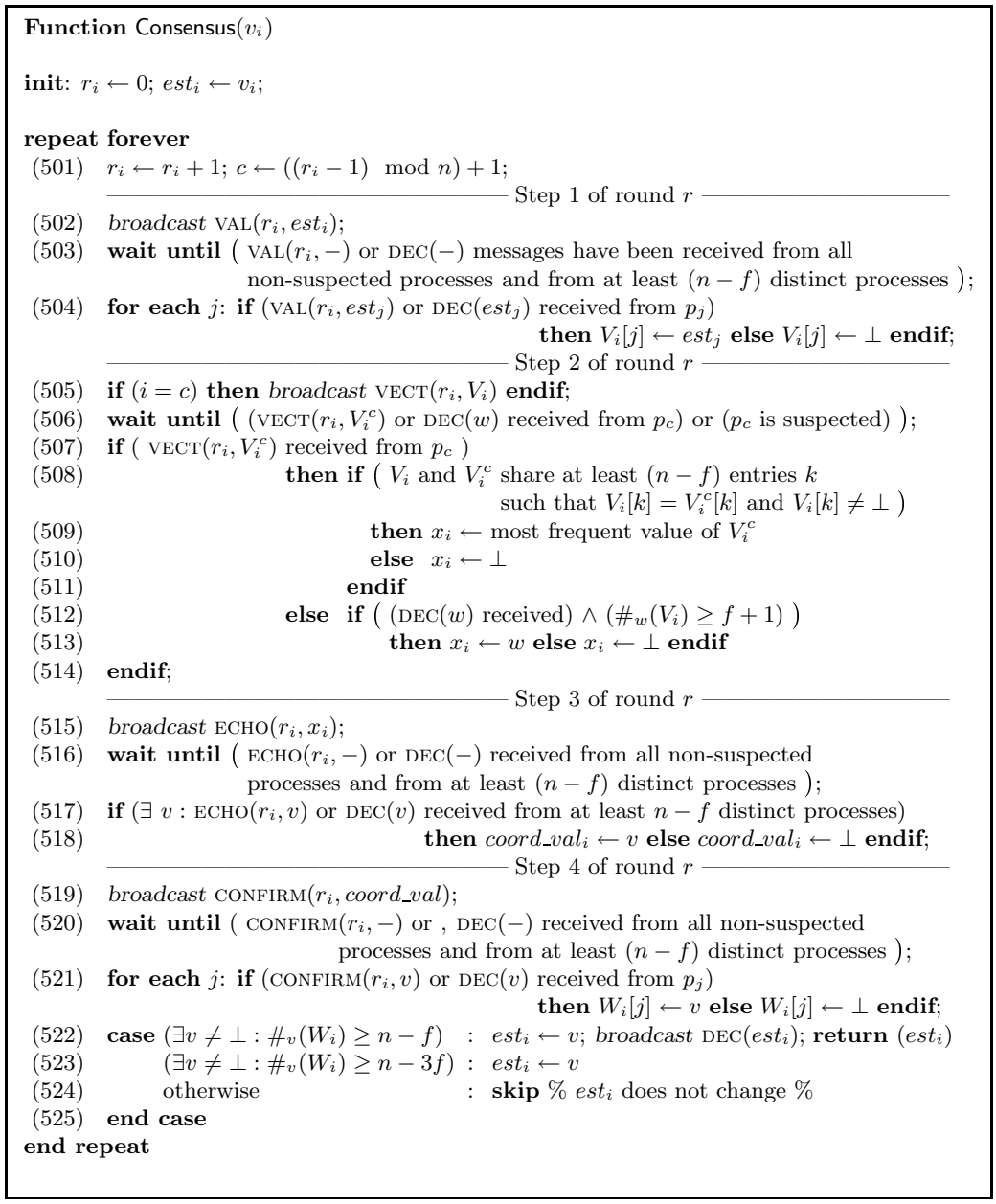

Figure 5: A $\diamond P_{mute}$-Based Protocol for $n > 4f$

Lemma 14 *Let us assume $n > 4f$, and consider the situation where, at the beginning of a round r, all the correct processes p_i have the same estimate value v (i.e., $est_i = v$). They will never change their estimates, thereafter.*

Proof Note that in round r, all correct processes have v as their initial estimate, each correct process collects at least $(n - f)$ estimates, and there are at most f Byzantine processes. Thus, the view V_i built by any correct process p_i during r (at line 504) contains at least $n - 2f$ entries (i.e., a majority) equal to v (plus possibly f entries equal to $\perp$, due to asynchrony in the waiting statement of line 503, and at most f entries equal to $v' \neq v$ due to Byzantine processes).
Claim. At the end of the second step of r, $x_i = v$ or $x_i = \perp$ for any correct p_i. *End of the claim.*
It follows from this claim that $x_i = v$ or $x_i = \perp$ when a correct process p_i enters the third step (line 515). As $n - f > f$, we conclude from the test at line 517 that the local variable $coord\_val_i$ of any correct process p_i can only be equal to v or $\perp$ (line 518). Therefore, when a correct process p_i executes the last step of the round r, its array W_i can only contain v, $\perp$, and no more than f occurrences of $v' \neq v$ (due to the Byzantine processes). As $n > 4f$, then there is no non-$\perp$ value $v' \neq v$ such that $\#_{v'}(W_i) \geq n - 3f$. It then follows from the case statement at lines 522-525 that est_i is either set to v or remains unchanged (i.e., remains equal to v). Consequently, the estimates of the correct processes remains equal to v.

Proof of the claim. We have to show that, at the end of the second step of r, $x_i = v$ or $x_i = \perp$ for any correct process p_i. We consider three cases.

- Let us first consider the case where p_i received a VECT(r_i, V_i^c) message from p_c.

  Observe that V_i contains at most f entries equal to $\perp$, at most f entries equal to $v' \neq v$ (due to Byzantine processes). All its other entries are equal to v (there are at least $(n - 2f)$ such entries).

  If V_i and V_i^c share at least $(n-f)$ non-$\perp$ entries (line 508), we can conclude from the previous observation on V_i that V_i^c has at least $(n - 2f)$ entries equal to v. As $n > 4f$, it follows that v is a majority value in V_i^c and consequently x_i is set to v at line 509. In the other case (p_i receives a vector from p_c and the test of line 508 is not satisfied), x_i is set to $\perp$ (line 510).

- Let us now consider the case where p_i received a DEC(w) message from p_c. If $\#_w(V_i) < (f+1)$, then p_i sets x_i to $\perp$, and the claim follows. If $\#_w(V_i) \geq (f + 1)$, then p_i sets x_i to w. We show that in that case $w = v$.

  As at most f processes are Byzantine, we conclude from $\#_w(V_i) \geq (f + 1)$ that at least one correct process had its estimate equal to w at the beginning of the round r. As all the correct processes have their estimates equal to v at the beginning of r (lemma assumption), $w = v$.

- Finally, if p_i suspects p_c, it sets x_i to $\perp$ (line 513).

End of the proof of the claim. $\square_{Lemma\ 14}$

Lemma 15 [Validity] *If all the correct processes propose the same value v, then no value $v' \neq v$ can be decided.*

Proof The proof of this lemma is an immediate consequence of Lemma 14. $\square_{Lemma\ 15}$

Lemma 16 [Agreement] *Let $n > 4f$. No two correct processes decide different values.*

Proof Let r be the first round during which a correct process p_i decides, and let v be the value it decides. The decision is obtained at line 522, which means that v is such that $\#_v(W_i) \geq n - f$. Let us consider the worst case, i.e., the case where p_i received $(n - f)$ messages CONFIRM(r, v) and f messages CONFIRM$(r, \perp)$. Hence, W_i contains exactly $(n - f)$ occurrences of v and f occurrences of $\perp$.

Let us now consider the vector W_j of any other correct process p_j. In the worst case, p_j received the f messages CONFIRM$(r, \perp)$ (the same as the ones received by p_i), does not receive f CONFIRM(r, v) messages because it suspects their senders, and receives also f CONFIRM(r, v') messages (with $v' \neq v$) send by f Byzantine processes. It follows that $\#_v(W_j) \geq n - 3f$. Moreover, as $n - 3f > f$, it follows that there is a single non-$\perp$ value (namely, v and not v') that satisfies $\#_v(W_j) \geq n - 3f$. Consequently, when any correct process executes r, it executes either Line 522 or Line 523, after which all correct processes have their estimates equal to v. Thus, if a correct process decides during r, it decides v (line 522). If it decides during a round $r' > r$, due to Lemma 14, it cannot decide a value different from v. $\square_{Lemma\ 16}$

Lemma 17 *No correct process can block forever in a round.*

Proof The proof of this lemma follows directly from the two following facts: (1) there are as most f faulty processes, and (2) the failure detector satisfies the Muteness Strong Completeness property, that ensures no process blocks forever in a **wait** statement. $\square_{Lemma\ 17}$

Lemma 18 [Termination] *Let $n > 4f$. Each correct process eventually decides.*

Proof Let t be the time after which the failure detector is accurate, i.e., no correct process is suspected (due to the Eventual Strong Accuracy of the failure detector, such a time t does exist). Let r be the first round that starts after t and is coordinated by a correct process p_c. Let us observe that, due to Lemma 17 and the use of DEC() messages (if any), any correct process p_i that has not yet decided starts round r.

Let us first observe that when the correct processes execute the first step of r, they get V_i vectors that differ in at most f entries (the entries corresponding to the Byzantine processes). So, when, during the second step, a correct process p_i compares V_i and V_i^c, the test of line 508 is satisfied, and consequently all the correct processes p_i set x_i to the most frequent value of $V_i^c = V_c$. Denote this value by v.

Then, during the third step of r, due to condition of the **wait** statement (after t no correct process is suspected), each correct process receives at least $(n - f)$ copies of v and adopts that value in $coord\_val_i$. Similarly, during the last step of r, each correct process receives at least $(n - f)$ CONFIRM(r, v) messages and decides at line 522. $\square_{Lemma\ 16}$

Theorem 7 *Let $n > 4f$. The protocol described in Figure 5 solves the Consensus problem.*

Proof The proof follows from Lemmas 15, 16 and 18. $\square_{Theorem\ 7}$

5 Conclusion

This paper has presented two families of protocols for solving Consensus in asynchronous distributed systems prone to Byzantine failures. In both families, the processes proceed by executing

asynchronous rounds. The protocol of the first family assumes a random oracle, while the other two protocols are based on an inaccurate failure detector oracle. All these protocols share the following noteworthy features:

- They rely on simple design principles and are relatively easy to understand.
- Their message complexity is $O(n^2)$ per round.
- In addition to the round number they carry, the size of any message is $O(1)$[5].
- No protocol requires "heavy" mechanisms such as "message proofs", certificates, or any kind of application level signatures.
- Each protocol ensures termination in one round in favorable circumstances. More, the random oracle-based protocol as well as one of the failure detector-based protocols achieve termination in a single communication step in these cases, which is time optimal.
- In all the protocols, a process uses a simple broadcast mechanism to prevent the other processes from deadlocking by informing them it has decided (sending of the DEC() messages). This broadcast is not reliable in the sense that only a subset of the processes can receive the DEC() messages if the sender fails while issuing the broadcast[6]. The way these messages are used to prevent deadlock is new, particularly simple and cheap.

The price that these protocols have to pay to enjoy the previous properties lies in the fact that they are not optimal with respect to the number of Byzantine failures that can be tolerated in an asynchronous distributed system (i.e., they do not meet the $n > 3f$ lower bound). Yet, given the reduced cost of hardware, it is interesting to consider employing more redundant hardware in favor of simpler and more efficient protocols. Moreover, the design of asynchronous Byzantine consensus protocols that meet all lower bounds (on the value of f, the size of messages, the early decision in good circumstances, etc.) is still an open problem.

A last remark concerns the flexibility of the proposed randomized consensus protocol, that allows the system designer to place the randomization assumption either in the protocol or in the network (see Appendix A). This protocol actually terminates with probability 1 if only one of the following holds: the underlying oracle is truly random (and not only pseudo-random), or the delivery order of messages by the network is random. Since its convergence is guaranteed if any one of the alternative assumptions is satisfied, when both assumptions can be satisfied, the proposed protocol has the nice feature that it provides increased overall assumption coverage for free [13].

References

[1] Ben-Or M., Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*, ACM Press, pp. 27-30, Montréal (Canada), 1983.

[2] Berman P. and Garay J.A., Cloture Voting: $n/4$-Resilient Distributed Consensus in $t + 1$ Rounds. *Mathematical System Theory*, 26(1):3-19, 1993.

[5] Except for the last protocol in which, at each round, there is one step in which *only* the round coordinator needs to send one vector to everyone, i.e., the size of such a message is $O(n)$.

[6] Differently, other failure detector-based asynchronous consensus protocols (e.g., [4, 9]) require a *reliable* broadcast primitive to attain the same goal. The message cost of such a primitive is $O(n^2)$.

[3] Cachin Ch., Kursawe K. and Shoup V., Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 123-132, Portland (OR), 2000.

[4] Chandra T.D. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.

[5] Doudou A., Garbinato B. and Guerraoui R., Encapsulating Failure Detection: from Crash to Byzantine Failures. *Proc. International Conference on Reliable Software Technologies*, Vienna (Austria), 2002.

[6] Doudou A., Garbinato B., Guerraoui R. and Schiper A., Muteness failure Detectors: Specification and Implementation. *Proc. 3rd European Dependable Computing Conference (EDCC'99)*, Springer-Verlag LNCS #1667, pp. 71-87, Prague (Czech Republic), 1999.

[7] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[8] Garay J.A. and Moses Y., Fully Polynomial Byzantine Agreement for $n > 3t$ Processes in $t+1$ Rounds. *SIAM Journal on Computing*, 27(1):247-290, 1998.

[9] Hurfin M., Mostefaoui A. and Raynal M., A Versatile Family of Consensus Protocols Based on Chandra-Toueg's Unreliable Failure Detectors. *IEEE TC*, 51(4):395-408, 2002.

[10] Knuth D.E., The Art of Computer Programming, Volume 1: Fundamental Algorithms. *Addison Wesley*, Reading (MA), First Edition 1968, Second Edition 1973.

[11] Lamport L., Shostak R. and Pease L., The Byzantine General Problem. *ACM Transactions on Prog. Languages and Systems*, 4(3):382-401, 1982.

[12] Pease L., Shostak R. and Lamport L., Reaching Agreement in Presence of Faults. *Journal of the ACM*, 27(2):228-234, 1980.

[13] Powell D., Failure Mode Assumptions and Assumption Coverage. *Proc. of the 22nd Int. Symp. on Fault-Tolerant Computing (FTCS-22)*, Boston, MA, pp.386-395, 1992.

[14] Rabin M., Randomized Byzantine Generals. *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, IEEE Computer Society Press, pp. 403-409, Tucson (AZ), 1983.

[15] Schneider F.B., Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299-319, 1990.

[16] Toueg S., Randomized Byzantine Agreement. *Proc. 3th ACM Symposium on Principles of Distributed Computing (PODC'84)*, ACM Press, pp. 163-178, Vancouver (Canada), 1984.

A Coin Randomness *vs* Power of Byzantine Processes

This appendix motivates why either (1) the coin has to be completely random and Byzantine processes have no access to the coin of round $r+1$ before their messages of round r have been received by all correct processes, *or* (2) the Byzantine processes have to be prevented from controling the order in which messages are delivered to the correct processes. Suppose, to the contrary, that the coin tosses are known in advance, as is the case when all nodes know the seed and employ the same pseudo-random number generator, *and* the Byzantine processes fully control the network. Also, fix $n = 5f + 1$. It is possible that in some execution, processes start a round such that the estimate of $2f + 1$ correct processes is v, the estimate of the other $2f$ correct processes is v', and there are f Byzantine processes, as depicted in Figure 6 (vector V in that figure). In particular, this could happen in the first round of the protocol. Moreover, the result of the coin toss for this round is v', which is the estimated value held by only $2f$ processes.

Next, we are going to show that under these assumptions, with our protocol, the Byzantine processes can intentionally control which processes adopt v and which adopt v' as their estimate for the next round. Specifically, they can force a process p_i to adopt v as following: all Byzantine processes send v to p_i, and ensure that the f values that p_i misses are v' (from correct processes) – this is the vector V_i in Figure 6. Similarly, to force a process p_j to adopt v', all Byzantine processes send v' to p_j, and ensure that the f values that p_j misses are v (from correct processes) – this is the vector V_j in Figure 6. Thus, according to the protocol described in Figure 1, p_i will adopt v, since v appears at least $n - 2f$ times in V_i, while p_j will adopt s_j, the coin toss, since no value appears in V_j more than $n - 2f$ times. Recall also that we assume that $s_j = v'$ in that round.

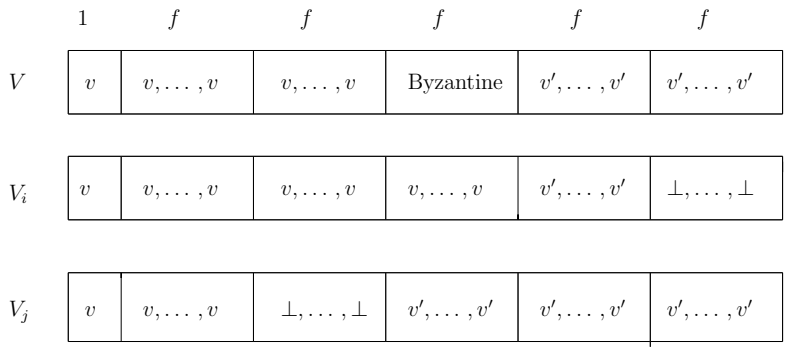

Figure 6: A Bad Scenario (that can occur when the coin is not truly random and the Byzantine processes control the delivery order of messages)

Observe that this scenario relies on the fact that the result of the coin toss was the same as the one supported by only $2f$ correct processes, and all other $2f + 1$ correct processes supported the opposite value, namely, v'. Thus, if (1) a round r starts in the situation described above, (2) the Byzantine processes know beforehand both values of the coin during r and $r + 1$, and (3) they control the network, then they can ensure that $2f$ processes that started with v keep v (e.g., p_i in the previous discussion, Figure 6), $2f$ processes that started with v' keep v' (e.g., p_j in the previous discussion, Figure 6), and one of the processes that started round r with v, will adopt the opposite value of the coin toss of round $r + 1$. Clearly, the same arguments, but with exchanging v and v', can be applied to the opposite case: $2f + 1$ correct processes start with v', $2f$ correct processes start with v, and the coin toss is v. Thus, if the Byzantine processes control the message delivery order, they can prevent the protocol from terminating.

So, in order to prevent the bad scenario described above, we need to assume that the Byzantine processes either cannot control the delivery order of messages by the network to the correct processes, or do not know the future coin tosses. In practice, the easiest way to implement a shared coin is by starting with a shared seed and all processes applying the same pseudo random number generator. If indeed Byzantine processes cannot control the network, as is likely to happen in many systems, then this means that we have a very practical solution for consensus. Moreover, if the delivery order of messages from the network to correct processes is somewhat random, then in fact we can invoke our protocol with always the same seed, which turns it into a deterministic protocol for solving consensus. We would like to emphasize that the termination of our protocol, when invoked with a deterministic oracle, depends only on the order of delivery of messages to correct processes, and not on the latencies of these messages. This highlights the flexibility of placing the randomness assumptions either in the protocol, or at the network level.