ASYNCHRONOUS BYZANTINE CONSENSUS[1]

Chagit Attiya
Danny Dolev
Joseph Gil

Institute of Mathematics and Computer Science
Hebrew University, Jerusalem

ABSTRACT: Reaching agreement in an asynchronous environment is essential to guarantee consistency in distributed data processing. All previous asynchronous protocols were either probabilistic or they assumed a fail–stop mode of failure. The deterministic protocol presented in this paper reaches a Strong Byzantine Agreement in a system of asynchronous processors; and therefore can sustain arbitrary faults. In our model, processors can be completely asynchronous, though the communication network has the property that a message being sent by a correctly operating processor to a set of processors will reach its destinations within a predetermined period Δ. Additional results presented in the paper prove that in the above model one cannot reach a consensus within a bounded time. A correctly operating processor should wait to receive messages from other processors before making a decision. This result holds also for Weak Byzantine Agreement, but not for nontrivial consensus. We present a trivial protocol to reach a nontrivial consensus in bounded time.

[1] The research was supported in part by the United States - Israel Binational Science Foundation, grant no. 2439/82.

1. Introduction

The problem of reaching agreement among independent processors is a fundamental problem of both practical and theoretical importance in the area of distributed systems; see, e.g. [Ag, BL, DRS, DSb, LSP, MSF, T]. We consider a system of n processors $p_1, \ldots, p_n$ ($n \geq 2$) that communicate by sending messages to each other. Initially, each p_i has a binary value x_i. At some point during its computation, a processor has to *decide* irreversibly on a binary value v. Each processor follows a deterministic protocol involving the reception and transmission of messages. Even though the protocols of individual processors are deterministic, there are several potential sources of nondeterminism in the system: processors might run at varying speeds, having received a set of messages a processor cannot determine the order in which they were sent, the behavior of faulty processors can be completely arbitrary. We allow Byzantine failures; that is, there is no assumption about the behavior of faulty processors.

A protocol reaches a *consensus* if:

(C1) no matter how the system runs, every nonfaulty processor makes a decision after a finite number of steps,

(C2) no matter how the system runs, two different nonfaulty processors never decide on different values.

A protocol reaches a *Nontrivial Byzantine Agreement* if it also satisfies:

(CN) 0 and 1 are both possible decision values for (possibly different) assignments of initial values.

(Condition (CN) is needed to avoid the trivial solution where each processor decides 1 regardless of its initial value. The decision value may depend only on the existence of faulty behavior).

A protocol reaches a *Weak Byzantine Agreement* if it also satisfies:

(CW) 0 or 1 is the decision value when all processors are nonfaulty and all have initial value 0 or 1, respectively.

A protocol reaches a *Strong Byzantine Agreement* if it also satisfies

(CS) 0 or 1 is the decision value when all correct processors have initial value 0 or 1, respectively.

M. Fischer [F] and L. Lamport [L] introduced the various agreements. Although the agreements are almost the same, the small differences among them distinguish between possibility and impossibility to reach consensus in certain situations.

If the processors and the communication system are completely reliable, consensus protocols trivially exist. The problem becomes interesting when the protocol must operate correctly in spite of the existence of faults in the system. The failure mode studied in [DDS] was *fail–stop*, thus a failed processor neither sends nor receives messages. The ability or impossibility to reach various agreements in the presence of Byzantine failures was left as an open question there. In this paper we resolve some of the open questions in [DDS]. A consensus protocol is *t-resilient* if it operates correctly when at most t processors fail. The existence of n-resilient consensus protocols when the processors and the communication system are both synchronous is known ([DFFLS,DSa,LSP,Rc]). Intuitively, synchronous processors implies that the internal clocks of the processors are synchronized to within some bounded rate of drift. Synchronous communication implies that there is a fixed upper bound on the time for a message to be delivered. These synchronizations are assumed in much of the research on "Byzantine Agreement" (see [DSc, F]).

In two recent papers ([FLP, DDS]) an extensive study of possibility and impossibility of reaching consensus indicates that in most asynchronous models it is impossible to reach consensus by deterministic protocols. If processors are asynchronous and the network is synchronous, then we have one of the few cases in which there exists a protocol for the fail–stop mode. In this paper we concentrate on this model, which is somewhat natural for a large distributed system of processors, since individual processors are asynchronous, that is, one cannot anticipate when a processor will respond to a message (a faulty one may never respond). This assumption relaxes the assumption of having an upper bound on the time at which a correct processor should respond to a given message. This assumption is natural when correctly operating systems tend to be saturated and operate very slowly. The communication network that we consider will be Δ–synchronous, i.e., a message sent by a nonfaulty processor to a set of processors will reach its destinations within Δ units of time.

Having assumed completely asynchronous processors, we have to define how time is measured. It is convenient to imagine that one is standing outside the system holding a "real time clock" that ticks at a constant rate. At each tick of the real clock, every processor can take at most one step. The processors are modeled as infinite–state machines. In the most general definition of "step", a processor can attempt to receive a message, and based on the value of the received message (or based on the fact that no message was received) it can change its state and broadcast a message to all processors. Processors need not have synchronous clocks or the capability to precisely measure Δ. The only requirement is the ability of every nonfaulty processor to measure Δ units of time on its timer that are not shorter that those of the "real time clock".

The two protocols to reach Strong Byzantine Agreement presented in the paper consist of two interleaved processes. One is the process of emulating phases. The processors run a protocol called the Phase Protocol, that enables active processors to synchronize themselves. The second process is simulation of the basic Byzantine Protocols ([DSa, DFFLS]) in order to agree on a set of values. The key point in the phase protocol is that when a processor realizes that other processors are running faster than it, it does not send new values it holds, even if these values are crucial and may influence the decision. The phase protocol can be used to control semi-synchronized behaviors in asynchronous systems.

The protocol with authentication presented operates in $t + 2$ artificial "phases", where each phase is composed of broadcasting a message and waiting to receive messages from at least $3t+1$ other processors. Therefore, the time it takes, for a given processor, to reach an agreement depends on the rate at which other correct processors proceed. The number of messages being sent during the protocol is $O(nt + t^3 \log t)$.

The protocol without authentication presented is based on the "*" protocol of [DFFLS]. It operates in $2t+4$ artificial phases, and as in the protocol with authentication, each phase is broadcasting and receiving from at least $4t+1$ other processors. The number of messages being sent is about the same as those in [DFFLS], $O(nt + t^3)$.

A processor reaches an agreement in *bounded* time if the number of steps it executes before reaching a decision state depends only on the number of processors and Δ, thus independent of the rate at which other processors proceed. The presented protocols do not have that property, while the one of Theorem E.1 in [DDS] does. We prove that even a Weak Byzantine Agreement cannot be reached in bounded time. On the other hand we present a protocol to reach Nontrivial Byzantine Agreement within bounded time. The Nontrivial Protocol ends with a decision value different from 0 only when all faulty processors send signed messages saying "I am faulty" early in the protocol. A Nontrivial Agreement may have a merit only in rare cases.

Notice that Rabin's protocol [Ra] for the choice coordination problem works also for asynchronous processors, and implicitly assumes synchronous communication. In that protocol, a processor may reach a decision within bounded time. But as in [DDS] the only failure mode the protocol sustains is fail-stop. We conjecture that the choice coordination problem cannot be solved in bounded time, in the presence of Byzantine Failure (the problem itself should also be redefined appropriately).

Another bound we obtain is the ratio between correct and faulty processors that has to hold in order to reach agreement. The protocol with authentication for Strong Byzantine Agreement requires $n > 4t$. If one assumes a stronger broadcasting model, then an agreement can be reached when $n > 3t$. We prove that the ratio $n > 3t$ is essential in both models in order to reach a Strong Byzantine Agreement. Note that the protocol without authentication requires $n > 5t$.

Some of the protocols presented assume the existence of an authentication scheme in which a faulty processor cannot forge a signature of other processors on a message and every processor can identify the signature of every other processor. One may use a scheme like [RSA], or even just some error correction (detection) code, depending on the degree of resiliency one wishes to tolerate.

2. Definitions

We follow the formalism of [DDS]. A *consensus protocol* is a system of n ($n \geq 2$) processors $P = p_1, \ldots, p_n$. The processors are modeled as infinite-state machines with state set Z. There are two special *initial states* z_0 and z_1. For $v = 0,1$, a processor

starts in state z_v if its initial value is v. Each processor then follows a deterministic protocol involving the reception and transmission of messages. The messages are drawn from an infinite set M. Each processor has a *buffer* for holding the messages that have been sent to it but which the processor has not yet received. Each buffer is modeled as an unordered set of messages (message order asynchronous, in the terminology of [DDS]). The collection of buffers support two operations:

Send(p,m): places message m in p's buffer;

Receive(p): deletes some collection (possibly empty) of messages from p's buffer and delivers these messages to p.

The protocol of each processor p is specified by a *state transition function* δ_p and a *sending function* β_p,

$$\delta_p : Z \times \tilde{M} \rightarrow Z,$$

$$\beta_p : Z \times \tilde{M} \rightarrow \{B \subseteq P \times M \mid B \text{ is finite}\},$$

where $\tilde{M}$ is the set of finite subsets of M. A pair (q,m) in the range of β_p means that p sends message m to processor q. Since we place no constraints on the message set M, we can assume that for each $p,q \in P$, $z \in Z$ and $\mu \in \tilde{M}$ there is at most one message m with $(q,m) \in \beta_p(z,\mu)$. It is also convenient to assume that a processor attaches its name and a sequence number to each message so that the same message m is never sent by two different processors nor at two different times.

Throughout the paper we assume that receive/send is *atomic*; thus, in any state in Z a processor can both receive and send messages. The operation of receiving, analyzing, and sending to a set of processors is one atomic indivisible step. Note that if the receive/send is not atomic, there is no way to reach a consensus with one or more faulty processors ([DDS]).

The transmission model assumed is *broadcast*: a processor can send messages to any set of processors in one step. In all the protocols presented a nonfaulty processor will send every message to all other processors (the terms "send" and "broadcast" are both used). However, we do not assume that faulty processors are forced to send every message to *all* processors. Note that in *point − to − point* transmission there is no way to reach consensus with two or more faulty processors [DDS].

A *configuration* C consists of

(i) n states $st(p_i,C) \in Z$, for $1 \leq i \leq n$, specifying the current state of each processor, and

(ii) n finite sets $buff(p_i,C) \in \tilde{M}$, for $1 \leq i \leq n$, specifying the current content of each buffer.

Initially, each state is either z_0 or z_1 as described above, and each buffer is empty.

An *event* is a pair (p,μ) where $p \in P$ and $\mu \subseteq \tilde{M}$. Think of the event (p,μ) as the reception of the set of messages μ by p. Processor p is said to be the *agent* of the event (p,μ). An event (p,μ) is *applicable* to configuration C only if $\mu \subseteq buff(p,C)$.

If the event $e = (p,\mu)$ is applicable to C, then the next configuration $e(C)$ is obtained as follows:

(a) p changes its state from $z = st(p,C)$ to $\delta_p(z,\mu)$ and the states of the other processors do not change,

(b) for all $(q,m) \in \beta_p(z,\mu)$, add m to $buff(q,C)$,

(c) delete μ from $buff(p,C)$.

To define "correctness" of a protocol, we must consider sequences of events. A *schedule* is a finite or infinite sequence of events. Each event is a step taken by some processor. For simplicity assume that each step takes at least one unit of "real time". Hence, a processor that takes one step, knows that at least one real time unit passed. A schedule $\sigma = \sigma_1, \sigma_2, \cdots$ is *applicable* to an initial configuration I if:

(1) the events of σ can be applied in turn starting from I, i.e., σ_1 is

applicable to I, σ_2 is applicable to $\sigma_1(I)$, etc.;

(2) for every j, let $\sigma_j = (p,\mu)$, if a message m was sent to p by an event σ_i with $i \leq j - \Delta$ (constant $\Delta \geq 1$) and if none of the events σ_k with $i < k < j$ is the reception of m by p, then m belongs to μ.

Condition (2) above is the Δ-synchronous communication requirement.

If σ is finite, $\sigma(I)$ denotes the resulting configuration, which is said to be *reachable* from I. A configuration reachable from some initial configuration is said to be *accessible*. Henceforth, all configurations mentioned are assumed to be accessible. If Q is a set of processors, the schedule σ is *Q-free* if no $p \in Q$ takes a step in σ. A schedule together with the associated sequence of configurations are called a *run*.

A processor p is *nonfaulty* if all of its steps are according to the transition functions δ_p and β_p, and in an infinite run it takes infinitely many steps. Processor is *faulty* otherwise. Processor p is nonfaulty in accessible configuration C if it is nonfaulty in a run that accesses C, and all its steps are according to δ_p and β_p.

We assume that there are two disjoint sets of *decision states* Y_0 and Y_1, such that if a processor enters a state in Y_v, $(v \in \{0,1\})$, then it must remain in states in Y_v. We say that a processor p *decided on* v in a configuration C if $st(p,C) \in Y_v$. A configuration C has *decision value* v if $st(p,C) \in Y_v$ for some nonfaulty processor p.

A consensus protocol is *partially correct* if:

(C2) no accessible configuration has more than one decision value, and

(CN) for each $v \in \{0,1\}$, some accessible configuration has decision value v.

For $0 \leq t \leq n$, an infinite run is a *t-admissible run from I* if:

(1) the associated schedule is applicable to I,

(2) at most t processors are faulty in it.

A run is *deciding* if every nonfaulty processor enters a decision state. A protocol is *t-resilient for the Nontrivial Byzantine Agreement problem* (NBA in short) if it is partially correct and

(C1) every t-admissible run from every initial configuration is a deciding run.

A protocol is *t-resilient for the Weak Byzantine Agreement problem* (WBA in short) if it also satisfies

(CW) if I_v is the initial configuration in which all processors have initial value v, then all configurations reachable from I_v by 0-admissible deciding runs have decision value v.

A protocol is *t-resilient for the Strong Byzantine Agreement problem* (SBA in short) if instead of (CW) it satisfies

(CS) if I_v' is the initial configuration in which all nonfaulty processors have initial value v, then all deciding configurations reachable from I_v' by t-admissible deciding runs have decision value v.

For our impossibility proofs we have to define the applicability of a schedule to a noninitial configuration C. If C is reached from initial configuration I by schedule τ, then σ is *applicable* to C iff $\tau\sigma$ is applicable to I. (To be completely precise we should include the history τ as part of the configuration C. However, in our impossibility proofs, the history is clear from the context, so we simply say that σ is applicable to C rather than to (C,τ).)

3. The Impossibility Result for a Bounded Protocol

Let Θ be a constant depending only on n, the number of processors, and Δ, the communication time bound. A protocol is said to be $\Theta-$*bounded* provided that for every processor p, every configuration C, and every schedule σ applicable to C, if p takes Θ steps in σ, then

$$st(p,\sigma(C)) \in Y_0 \bigcup Y_1.$$

Notice that the protocol of Theorem E.1 in [DDS] is a bounded protocol.

Definition. Let $X \subset P$. Two configurations C and D are $X-$*equivalent* if, for every $p \in P-X$, $st(p,C) = st(p,D)$ and $buff(p,C) = buff(p,D)$.

Definition. Let $e=(p,\mu)$ be an event applicable to a configuration C. Event e (when applied to C) is a *total reception* if $\mu = buff(p,C)$.

Our impossibility result assumes a total reception rather than $\Delta-$synchronous communication, and therefore of a stronger nature.

Lemma 1. Let D and D' be two configurations, and let p and q be two processors, such that p is nonfaulty at D deciding 1 in it, and q is nonfaulty at D' deciding 0 in it. . In any $\Theta-$bounded $1-$resilient consensus protocol, configurations D and D' are not $\{p,q\}-$equivalent.

Proof. Assume on the contrary that the conditions of the Lemma hold and for some $\Theta-$bounded $1-$resilient consensus protocol, D and D' are $\{p,q\}-$equivalent. Let σ be a $\{p,q\}-$free schedule in which every processor in $P-\{p,q\}$ makes Θ steps, where all the steps are total reception. Schedule σ is applicable to both D and D'. Since the protocol is $\Theta-$bounded, all nonfaulty processors have to be at a decision state by the end of the run, whether the run is σ applied to D, or σ applied to D'. Moreover, in both cases, all processors reach the same decision value; assume w.l.o.g. that it is 0. But this decision violates requirement C2, because p is nonfaulty in D and decides on 1. □

In a $\Theta-$bounded consensus protocol a nonfaulty processor should enter a decision state within Θ steps, even when it is the only processor currently running. This implies that in a Weak Byzantine protocol a processor should decide on its own value. This observation leads to a contradiction.

Theorem 1. In our asynchronous model, there is no $1-$resilient $\Theta-$bounded Weak Byzantine Agreement protocol.

Proof: Assume on the contrary that such a protocol does exist. To obtain a contradiction we make use of a Byzantine behavior in which a faulty processor just ignores some of the incoming messages.

Denote by σ_p a schedule composed of Θ consecutive steps of p alone. The schedule σ_p is applicable to every initial configuration. Let I be an initial configuration with $st(p,i)=z_v$, then the $\Theta-$bounded condition implies that $st(p,\sigma_p(I)) \in Y_v$; that is, p decides on its initial value.

Let I be an initial configuration in which: p has initial value 1, another processor, q, has initial value 0, and all other processors have arbitrary values. Let $\sigma=\sigma_p\sigma_q$ and $\sigma'=\sigma_q\sigma_p$. In both schedules applied to I the first processor decides on its initial value. Define the following runs for σ and σ': processors q in σ and p in σ' are faulty and ignore all messages from other processors. Therefore q in σ behaves exactly as in σ', and p in σ' behaves exactly as in σ.

Define state $D=\sigma_p\sigma_q(I)$, and state $D'=\sigma_q\sigma_p(I)$. It is clear that D and D' are

$\{p,q\}$-equivalent. However, in D processor p is nonfaulty and decides on 1, whereas processor q is nonfaulty in D' and decides on 0; a contradiction to Lemma 1. □

4. Unbounded Protocols For Strong Byzantine Agreement

In the previous section we proved that in our asynchronous model even a Weak Byzantine Agreement cannot be achieved within bounded time. We show that without this requirement there are protocols to reach Strong Byzantine Agreement.

In the synchronous case, a protocol to reach SBA using authentication is simple (cf. [DSa]). The difficulties in using it as is arise from the lack of synchronism; moreover, even when one finds a way to introduce some notion of phase, a processor that is relatively slow and that is not active at a given phase, has to be provided with means to catch up with faster nonfaulty processors. The lower bound proof of the previous section indicates some of the problems we have to overcome.

4.1. The Phase Protocol

The basic idea in the Phase Protocol is somewhat similar to the idea of Protocol E.1 in [DDS]. Every processor sends a claim to be at phase k if it has received more than $2t$ claims about phase $k-1$, or more than t claims about phase k. Phases will be separated by 2Δ consecutive steps during which a processor does not receive any valid message. This phase separation enables us to guarantee that two correct processors will not send claims about two different phases "concurrently" (i.e., in less than Δ units of time apart). We assume that the communication system is such that the sender of a message can be verified.

We describe how to run the phase protocol in order to obtain K phases, for some constant K. The protocol uses two thresholds, TRH1 and TRH2, such that $TRH1 \geq TRH2 + t \geq 2t+1$.

We denote by $\langle m \rangle k$ a message sent by processor p_k. A $\varphi_i-claim$ by processor p_k is a message $\langle i \rangle k$. Processor p_k may send at most one φ_i-claim, for every i; denote this message by $\varphi_i(p_k)$. When a nonfaulty processor sends a message it broadcasts it to all processors, including itself.

Definition: A $\varphi_i(p_k)-$message is *valid* if it is the first one received from p_k about phase i, and $i \leq K$. A φ_i-claim is *valid* if it was received by a valid message and for $i > 0$ there are at least TRH2 valid $\varphi_{i-1}-$claims.

A *mute* step is a step during which a processor does not receive any valid $\varphi-$message.

The Phase Protocol

(1) Set phase counter PC to 0. Send a φ_0-claim.

(2) While $PC \leq K$ do:
wait for 2Δ consecutive mute steps, and then,

(2a) Let j, $j > PC$, be the maximal phase for which a valid φ_j-claim exists. If you have received at least TRH2 valid φ_j-claims, then broadcast a φ_j-claim to all the processors. Set PC to j and return to step (2).

(2b) Else, if you have received at least TRH1 valid φ_i-claims, where $i = PC$, then broadcast a $\varphi_{i+1}-$claim to all the processors. Increase PC by one and return to step (2).

(2c) Else, return to step (2).

(3) Stop.

A processor that sends a φ_i-claim as result of step (2b) of its protocol is called

active at phase i. No processor is active at phase 0.

Note that our model assumes that a processor can receive, decide what to send, and broadcast its messages in a single step. This assumption is crucial to the correction of the protocol. This assumption means that the last mute step in (2) and all subsequent (2a) to (2c) are being executed in a single step of a nonfaulty processor.

It is easy to see that if the number of faulty processors is bounded by t and $n \geq TRH1+t$, then the protocol terminates within a finite time. By simple induction on the phase number it can be shown that the phases do increase. The following theorems prove the basic properties of the Phase Protocol.

Theorem 2. Let $n \geq TRH1+t$. If an active nonfaulty processor p_k sends φ_i at "real time" T (as a result of step (2b) of its protocol), and an active nonfaulty processor p_l sends φ_j, for $j < i$, at "real time" T' (as a result of step (2b) of its protocol), then $T' < T-\Delta$.

Proof: Let us look at the real times at which the various steps of the processors occur. It is enough to prove the Theorem for $j = i-1$. For sending $\varphi_i(p_k)$, an active processor p_k should have received valid φ_{i-1}-claims from at least $TRH1-t$ nonfaulty processors. Since processor p_l waited for 2Δ mute steps before sending its claim as a result of step (2b) of its protocol, all the above claims should arrive to it not later than time $T-\Delta$. On the other hand, if processor p_l would have received these claims before time T' it would receive at least $TRH1-t \geq TRH2$ such claims and would use step (2a) instead of step (2b), which proves the theorem. □

Corollary 1. For each phase i, $1 \leq i \leq K-1$, all claims sent by nonfaulty processors active at phase i will be received by all processors active at phase $i+1$.

Corollary 2. For each phase i, $1 \leq i \leq K$, all claims sent by nonfaulty processors active at phase i will be received as valid claims by every nonfaulty processor.

Theorem 3. Let $n > TRH1+t$. For every phase i, $1 \leq i \leq K$, at least $TRH2-t$ nonfaulty processors are active at phase i.

Proof: The proof follows easily from the bound TRH2 at step (2a). □

The exact thresholds of the Phase Protocol will be chosen according to the extra restrictions about the ratio between faulty to nonfaulty processors arise from protocols that use the Phase Protocol, as we will see in the rest of the section.

4.2. Byzantine Protocol using authentication

We describe now how to use the Phase Protocol in order to run a Byzantine Agreement protocol while using authentication.

The protocol we present assumes the existence of some authentication scheme having the following properties:

(1) No processor can forge any signature (even a faulty one cannot);

(2) No processor can alter the content or the signature of a signed message undetectably.

(3) Every processor can identify a correct signature of every other processor.

Every processor will follow the protocol to know when, if at all, it is allowed to send new values it obtains. While exchanging the messages of the Phase Protocol the processors append to them, sometimes, proofs on new values they have obtained.

The protocol presented here requires $t+1$ nonfaulty processors active in each phase, hence by Theorem 3, it is enough to choose $TRH2 = 2t+1$, $TRH1 = 3t+1$ and $n \geq 4t+1$. A ratio $n \geq 3t+1$ can be

reached using only one nonfaulty processor active in each phase. This can be done assuming that every message received by a nonfaulty processor will be received by all nonfaulty processors within Δ. We call such communication network a *Strong Broadcast Network*. This requirement is too strong for a natural communication network and will not be used in the paper.

Definition: An α_0-message is a message $\langle v,p_k\rangle$, containing an information v of some processor p_k. The pair $\langle v,p\rangle$ is the value carried by the message, and p is the *subject* of the message. An α_i-message is an α_0-message followed by i distinct signatures $x_1, \ldots, x_i$, where the subject of the message is the first of them. (i.e. an α_i-message contains exactly i distinct signatures).

A correct processor will append a valid α_i-message to a φ_i-claim only when it is active, that is, it sends the claim as a result of step (2b) of its phase protocol. To simplify the presentation of the protocol assume that a processor receives its own α_0-message during phase 0 of the Phase Protocol.

The Byzantine Protocol

(1) Run the Phase Protocol with $K=t+2$.

(1a) If you are not active, act as in the Phase Protocol.

(1b) For every $1\leq i\leq K-1$, if you are active at phase i and you have received an $\alpha_{i-1}-$message carrying a value you have not sent before, add your signature to it and append it to the φ_i-claim.

(1c) Accumulate all the messages you receive during the protocol.

(2) At the end of the Phase Protocol, prepare a set Σ of values. Insert to this set every value on which you have received, in maybe different messages, at least $t+1$ distinct signatures, including the authentication (signature) of the subject of the message.

(3) Apply a decision function to Σ; for example: if Σ contains values $\langle 1,p\rangle$, for at least $t+1$ processors p, decide on 1; otherwise decide on 0.

Lemma 2: If a nonfaulty processor sends an α_i-message, $1\leq i\leq t$, then at least $t+1$ nonfaulty processors will send the value of that message in some α_j-message $(1\leq j)$.

Proof: (sketch) By Theorem 2, if a nonfaulty processor sends a φ_i-claim at step (2b) of its phase protocol, it will arrive at every other nonfaulty processor before that processor sends a $\varphi_{i+1}-$claim. By Theorem 3, there will be at least $t+1$ nonfaulty processors who will send a $\varphi_{i+1}-$claim as a result of step (2b) of their phase protocol. The α_i-message sent by p at phase i will reach every one of them. Everyone of these processors will append its signature and will send the value carried by the α_i-message, unless it has done so in some previous phase. □

Lemma 3: Every nonfaulty processor has the same set Σ.

Proof: Let α_0 be a value in the set Σ of some nonfaulty processor p_k. By the definition of Σ, p_k gathered $t+1$ distinct signatures on α_0. Let p_l be the first nonfaulty processor that signed a message containing α_0, say at phase j. If $j<t+1$, then, by Lemma 2, α_0 would be sent by at least $t+1$ nonfaulty processors to every processor, and therefore α_0 will be in the set of values of every nonfaulty processor. If $j=t+1$ then α_0 was received as $\alpha_{t+1}-$message by p_l. The $t+1$ signatures in it should include one of a nonfaulty processor beside p_l, a contradiction to the minimality property defined for p_l. □

Theorem 4. Let $n>4t$. The Byzantine Protocol reaches Strong Byzantine Agreements.

Proof: By Lemma 3, all nonfaulty processors apply the decision function on the same set of values. We have to show that if all nonfaulty processors have initial value 0, then the decision is 0, and if all have initial value

1, the decision is 1. By Theorem 3, at least $t+1$ nonfaulty processors are sending their own values at phase 1, and by Lemma 2, all these values will be in Σ. Therefore, if all nonfaulty starts with 0 (resp. 1), then the decision should be on 0 (resp. 1). □

We conclude this section by some complementary remarks about the protocol. The protocol actually runs in $t+2$ phases, because phase $t+3$ is required only to find out that phase $t+2$ was completed. An extra phase can be saved by combining phase 0 phase 1. The protocol is written for only two possible values, 0 and 1. It is easy to generalize it to any set of values. Moreover, one can add a preliminary phase during which every processor should try to get $n-t$ signatures on its value. This method prevents a faulty processor from introducing too many values during the protocol.

The total number of messages being sent during the protocol is $O(tn^2)$, where the number of values being sent is $O(n^3)$, because in the worst case all processors are synchronized, and everyone sends a value to everyone else. The number of different values each processor sends is $O(n)$. One can reduce the number of messages in our protocol to $O(nt + t^3)$ by using the same trick as in [DSa]; that is, by limiting the number of "active" processors and preventing the rest from taking an active part in the protocol. But in the asynchronous case it is better to pay the extra number of possible messages, and to be able to continue running the protocol whenever some subset of the processors is operating correctly, thus running the protocol at the rate of the $3t+1$ fastest nonfaulty processors.

4.3. Byzantine Protocol without using authentication

The protocol described here is based on the "* algorithm" ([DFFLS]) and assumes initial values of either 0 or 1 for each processor. As in the authenticated case, we modify a synchronous algorithm in order to run it in an asynchronous system using the Phase Protocol. Some other synchronous algorithms for reaching Byzantine Agreement can be similarly adapted. We could not adapt all the known algorithms, which suggests the question: what properties a synchronous algorithm should have in order to be adoptable to asynchronous system using the Phase Protocol?

We describe the algorithm rather informally and proofs are omitted. The reader is referred to [DFFLS] to complete the details. We will use the Phase Protocol starting at phase 1, for being consistent with [DFFLS].

The protocol uses two thresholds, $HIGH = 2t+1$ and $LOW = t+1$. When you have HIGH evidences to some information you know that every other nonfaulty processor has at least LOW evidences to the same information. The protocol requires HIGH nonfaulty processors active in each phase, therefore the ratio needed (according to Theorem 3) is $n \geq 5t+1$.

During the algorithm two types of messages will be sent: a "*" message and messages consisting of some name of some processor. The "*" represent the assertion that the sender has value 1, and a name represent that the named processor has sent "*".

Each processor keeps a record of all messages it has received. Consider this collection as held by some processor p. For the sake of simplicity we remove the index p from the sets defined below. Denote by W_x the set of processors that have sent the message x to processor p. We call W_x the set of *witnesses* to message x. Processor p is a *direct supporter* for processor r if p receives "*" directly from r. Processor p is an *indirect supporter* for r if it has a set of witnesses to r of cardinality LOW, i.e. if $|W_r| \geq LOW$ for p.

Processor p *confirms* r if the cardinality of the set of witnesses to r (i.e. $|W_r|$) is at least HIGH. Each processor p has a set (possibly empty) of confirmed processor

which we denote by C.

The last notion we need is "initiation", which means being ready to send "*". Processor p is *initiated* at phase k if either $k=1$ and at phase 1 it is active and has value 1, or at the beginning of phase k the cardinality of the set C of confirmed processors it has received is at least $LOW+\max(0,\lfloor k/2 \rfloor -1)$.

We assume that whenever a processor broadcasts a message to all others, it also sends one to itself, for purposes of recording its own messages. Every message is sent only once by a correct processor. Only active processors send messages. All processors collect messages and perform the other transitions (e.g. computing W_x). For technical reasons, and for being able to present our arguments about phases we assume that a processor processes its incoming messages (that are not messages of the Phase Protocol) just before sending a φ-claim at either step (2a) or (2b) of its Phase Protocol. Thus, when a processor notices that a phase has been just changed it reads its mail to find out what messages it have received since the last time he processed its messages.

We now give the rules for correct operation:

(R0) Run the Phase Protocol with $K=2t+5$ and with thresholds $TRH1=4t+1$ and $TRH2=3t+1$.

(R1) At phase 1 every active processor broadcasts "*" to all processors, if its value is 1.

(R2) If a processor is active at phase $k>1$ it broadcasts the names of all processors for which it is either direct or indirect supporter. If, in addition, it is initiated at this phase it also sends a "*" message.

(R3) If a processor confirms HIGH processors it *commits* to 1.

(R4) If, after phase $2t+4$, value 1 is committed then agree on 1; otherwise, agree on 0.

Note that processors may skip phases, when they are not active. A nonfulty nonactive processor keeps track of phases by scanning its incoming messages. Whenever it has TRH2 valid φ_i−claims, it knows that phase i has passed.

Let $d = 2t+4$. The following claims prove that the Protocol reaches SBA.

(*) If a nonfaulty processor p sends "*" at phase k, $1 \leq k \leq d-1$, then all nonfaulty processors will confirm p by phase $k+2$.

(*) If HIGH nonfaulty processors send "*" by phase k, $1 \leq k \leq d-2$, then by phase $k+2$ all nonfaulty processors commit.

(*) If p, q, r are nonfaulty processors, r active at phase k, $1 \leq k \leq d+1$, and x a message, then $r \in W_x$ of p by phase k iff $r \in W_x$ of q by phase k.

(*) If a nonfaulty processor r confirms p at phase k, $1 \leq k \leq d$, then all nonfaulty processors will confirm p by phase $k+1$.

(*) If HIGH nonfaulty processors are initiated at phase 1, then all nonfaulty processors commit by phase 3.

(*) If LOW nonfaulty processors send "*" by phase k, $1 \leq k \leq d-3$, then every nonfaulty processor commits by phase $k+4$.

(*) If a nonfaulty processor p does not send "*" by phase k, $1 \leq k \leq d$, then a nonfaulty processor q will neither be a direct support for it by phase $k+1$ nor an indirect support by phase $k+2$.

(*) If a nonfaulty processor commits at phase k, $2 \leq k$ then there are at least LOW nonfaulty processors which have sent "*" by phase $k-1$.

(*) If a nonfaulty processor commits by phase $d+1$, then all do.

(*) If only faulty processors initiate at phase 1, then no nonfaulty processor will ever commit.

Note again that for completing the details of the proof the reader has to read first the complete proof in [DFFLS], and then to fill in the details about the phases defined by the phase protocol. The main differences between our proof and the one in [DFFLS] are: in our protocol there is no single transmitter who sends its value at phase 1; at every phase, only HIGH nonfaulty processors necessarily sends messages; and the beginning and the end of a phase is more delicate in the our protocol. The number of bits exchanged in the above protocol (not counting the messages of the Phase Protocol) is as those of the algorithm in [DFFLS], and a similar trick as the active and passive processors can be applied here for further saving of messages.

5. A Trivial Bounded Protocol for Nontrivial Agreement

In Section 3, we have shown that Weak Byzantine Agreement cannot be achieved within bounded time. We contrast this result with the fact that a nontrivial agreement can be achieved within bounded time.

The following protocol obtains a consensus on 1 only in the strange case when all faulty processors "admit" to be faulty early in the protocol. In all the other cases the consensus is on 0.

All messages sent are signed. We assume an authentication scheme like the one used in Section 4. We use three types of messages:

(IMF) "I am faulty" message, may be sent only by faulty processors.

(D0) "decide 0" message.

(D1) "decide 1" message, always accompanied by IMF messages from t different processors, other than the sender.

During the protocol any nonfaulty processor sends at most one message of each type. Nonfaulty processors ignore a message if it is not of one of the above types, or if a message of the same type was already received from the same processor.

Agreement Protocol:

(1) Attempt to receive messages for 2Δ steps.

(2) If no valid message had arrived broadcast D0, decide on 0 and stop.

(3) If a valid D1 message was accepted, then decide 1 and stop.

(4) If there are t different IMF messages, then -

a. Send a D1 message.

b. Attempt to receive messages from nonfaulty processors for 2Δ steps. (By now the faulty processors are known).

c. If you have received a D0 message - decide 0, otherwise - decide 1.

d. Stop.

(5) Otherwise, return to step (1).

The above protocol exposes the weakness of the nontrivial agreement.

Theorem 5. For $n \geq t$ and $\Delta-$synchronous communication, the above protocol reaches a Nontrivial Byzantine agreement.

Proof. It is obvious that the protocol satisfies conditions C1 and CN. It remains to show that the protocol satisfies condition C2; that is, there is no run with two different decision values.

Assume on the contrary that there is such a run. Nonfaulty processors never send an IMF message, and a valid D1 message should be sent by a processor which did not send an IMF message. Hence, a faulty processor cannot create a valid D1 message.

If no nonfaulty processor sent a D1 message, then C2 is satisfied. It can be seen that a D0 message sent by a faulty processor will never change a decision of a nonfaulty processor.

The only case left is when some nonfaulty processor p broadcasts D1 and another one, say q, broadcasts D0. Let t_p, and t_q be the "real times" at which the processors send these messages, respectively. Assume on the contrary that they decide on 1 and 0, respectively. The $\Delta-$synchronous condition and step (4.b) of the protocol imply

$$t_p + 2\Delta < t_q + \Delta \qquad (1)$$

On the other hand, the $\Delta-$synchronous condition and step (3) implies that

$$t_q < t_p + \Delta \qquad (2)$$

Inequality (1) contradicts (2). □

Note the triviality of the protocol; we do not claim that this algorithm is of any use. It shows however, that the dependency of the agreement on the initial values disables reaching agreement within bounded time.

6. A Lower Bound on the Number of Faulty Processors

The protocol with authentication presented in Section 4 reaches consensus only for $n > 4t$. The protocol without authentication requires $n > 5t$. In this section we show that in our model $n > 3t$ is a necessary condition to achieve Strong Byzantine Agreement. There still remains to establish whether there is an protocol (using authentication and without assuming a strong broadcast) with $3t < n \leq 4t$, or whether $4t$ is a tight lower bound on the number of faulty processors such a protocol can handle. Similarly, what is the actual lower bound for protocols that so not use authentication?

Theorem 6. In a system of n asynchronous processors there is no t−resilient Strong Byzantine Agreement protocol, with $n > 3t$.

Proof. In any consensus protocol a decision must be reached after $n-t$ nonfaulty processors exchanged a finite number of messages. Therefore, for every initial configuration I and $X \subset P$, where $|X| \leq t$, if a schedule σ is X-free and applicable to I and all processors in $P-X$ are nonfaulty, then there is some finite prefix τ of σ, such that

$$st(p,\tau(I)) \in Y_0 \bigcup Y_1,$$

for all $p \in P-X$.

Assume that $n = 3t$. Partition the set of processors into 3 groups, A, B, C, each with t processors in it. Let I be the initial configuration in which processors in A are nonfaulty and have initial value 0, processors in B are nonfaulty and have initial value 1, and processors in C are faulty and have initial value 1.

Apply some $C-$free infinite schedule τ to I let τ' be its finite prefix in which all processors enter decision states. The protocol must reach Strong Byzantine Agreement. Therefore all decide on the same value, say w.l.o.g. 0.

Now look at the initial configuration I' similar to I in which processors in A are faulty and have initial value 1. Processors in C are nonfaulty and have initial value 1. Using τ' applied to I' define a run in which the faulty processors in A behave as though they were running from initial configuration I. Processors in group B receive the same messages in $\tau'(I')$ as in $\tau'(I)$. Hence, $\tau'(I')$ has decision value 0. □

Theorem 6 holds also for Strong Broadcast Network. Therefore, in this type of network the protocol of Section 4 is optimal.

References.

[Ag] H. Aghili, M. Astrahan, S. Finkelstein, W. Kim, J. McPherson, M. Schkolnick and H. R. Strong, A Prototype for a Highly available Database System, IBM Research Report RJ3755, 1983.

[BL] H. Breitwieser and M. Leszak, Improving availability of Partily Redundant Databases by Majority Consensus Protocols, in Distributed Database, H.J. Schneider (ed), North−Holland, 1982.

[DDS] D. Dolev, S. Dwork and L. Stockmeyer, On the Minimal Synchronism Needed for Distributed Consensus, proceedings, the 24th Annual Symposium on Foundations of Computer Science, 1983.

[DFFLS] D. Dolev, M. Fischer, R. Fowler, N. Lynch and R. Strong, Efficient Byzantine Agreement Without Authentication, Information and Control 3(1983), pp. 257-274.

[DRS] D. Dolev, R. Reischuk, and R. H. Strong, proceedings, the 23rd Annual Symposium on Foundations of Computer Science, 1982.

[DSa] D. Dolev and H. R. Strong, Authenticated Algorithms for Byzantine Agreement, Siam Journal on Computing 12(1983), pp. 656-666.

[DSb] D. Dolev and H. R. Strong, Distributed Commit with Bounded Waiting, Proceedings, Second Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, July 1982.

[DSc] D. Dolev and H. R. Strong, Byzantine Agreements, proceedings, COMPCON83, San Francisco, pp. 77-81, Mar 1983.

[F] M. J. Fischer, The Consensus Problem in Unreliable Distributed Systems, Proc. International Conference on Foundations of Computation Theory, 1983.

[FLP] M. J. Fischer, N. A. Lynch and M. S. Peterson, Impossibility of Distributed Consensus with one Faulty Process, Proc. 2nd ACM symp. on Principles of Database Systems, 1983.

[L] L. Lamport, The Weak Byzantine Generals Problem, JACM 30(1983), pp. 668-676.

[LSP] L. Lamport, R. Shostak and M. Pease, The Byzantine Generals Problem, ACM Trans. on Programming Languages and Systems 4(1982), pp. 382-401.

[MSF] Mohan, Strong and Finkelstein, Method for Distributed Transaction Commit and Recovery Using Byzantine Agreement within Clusters of Processors, Proc. ACM 2nd symp. on Principles of Distributed

Computing, 1983, pp. 89-103.

[Ra] M. O. Rabin, The Choice Coordination Problem, Acta Informatica 17 (1982), pp. 121-134.

[Re] R. Reischuk, A New Solution for the Byzantine Generals Problem, IBM Report RJ3673, November 1982.

[RSA] L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Comm. ACM 21(1978), pp. 120-126.

[T] R. H. Thomas, A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases, ACM TODS 4(1979) 2, pp. 10-209.