Asynchronous Approximate Agreement

A. D. Fekete

Department of Mathematics

Harvard University

Cambridge, MA 02138

fekete@huma1.harvard.edu

ABSTRACT: This paper introduces an algorithm to solve the Approximate Agreement Problem in an asynchronous failure-by-omission (or crash-failure) system, and proves that the algorithm is optimal by considering the power of the "adversary" scheduler to disrupt processors' views. We show that the adversary need not cause any crashes or omissions to achieve its purpose, and therefore no algorithm can do better than simply to operate round-by-round, as ours does. The resulting understanding of the adversary should be applicable to other problems in asynchronous failure-by-omission or crash-failure systems.

1 Introduction

A fundamental problem in designing fault-tolerant distributed systems is how to eliminate or reduce differences between the information held by different processors. A classical abstract version of this is known as the Byzantine Agreement Problem [PSL], and has been studied extensively, using many models of computation, reflecting differing amounts of synchrony in the system, different degrees of maliciousness on the part of faulty processors, different power of computation of processors, and different requirements on the solution (see [Fi] for a survey of these results). The distressing fact that in a system with asynchronous communication (i.e. where messages can take arbitrarily long to arrive) there is no agreement protocol that can tolerate even one fault, was first proved in [FLP], and extended to more general system models in [DDS].

Since reaching agreement is difficult even in synchronous systems, and impossible in asynchronous ones, several researchers have been led to study problems of reducing (rather than completely eliminating) differences between values held by processors. Obvious examples of such problems are clock synchronization ([LM], [LL], [HSSD]) and approximating a true value (e.g. a sensor) [MS]. An abstract formulation of the problem, which permits the use of techniques developed in studying Byzantine Agreement, is Approximate Agreement, introduced in [DLPSW], where algorithms were given for both synchronous and asynchronous systems assuming Byzantine (i.e. arbitrarily malicious) behaviour of faulty processors. Those algorithms proceed in rounds, where in each round each processor receives the current value held by other processors and "averages" these values to obtain a new value for itself (the function used is not the mean, but a fault-tolerant measure of central tendency). In [DLPSW] the algorithms given are shown to be optimal (for Byzantine faults) among algorithms having the same form, that is, where the value chosen in each round depends only on

values held by processors at the start of that round. The question is raised in [DLPSW], whether using information from other rounds permits better algorithms. This paper answers that question negatively for asynchronous systems in which processor failures are relatively benign (failure-by-omission), by giving an algorithm that proceeds in rounds and showing that this is optimal among deterministic algorithms.

The Approximate Agreement Problem is studied in this paper in the following form: there are n processors labelled 1,2,...,n which are linked by a completely connected, fault-free, point-to-point network which is the only means of interprocess communication. A message submitted to the network will eventually reach its destination (where it will be delivered if the addressee asks to receive it), but no upper bound exists on the time from source to destination. We are interested in protocols that are resilient to t failures, so we consider only executions where at least $n-t$ of the processors are correct, that is, they follow the given algorithm, and the remaining processors (the "faulty" ones, numbering at most t) are obliged to follow the protocol as well, except that they may neglect to send some messages the algorithm requires, and they may halt ("crash") at any time, without other processors being aware of the crash.[1] In each execution each processor p begins with an initial value, a real number $v(p)$, and (unless it has crashed) it must eventually enter a decision state with a new value $w(p)$ satisfying the validity condition that $w(p)$ lies within the interval of the initial values.[2] We will measure the performance of such an algorithm by K, the ratio of the size of the interval containing the new values to the size of the interval containing the initial values[3] (so a good algorithm is one with a low K).

The results of [DLPSW] indicate that any value for K can be achieved (so long as $n > 5t$) if enough communication is used, so we will restrict our discussion to algorithms using at most S rounds of communication.

The algorithm given in [DLPSW] has performance

$$K \leq \left\lceil \frac{n-3t}{2t} \right\rceil^{-S}$$

The algorithm given in this paper is very similar to that of [DLPSW], but it is valid whenever $n > t$, and it is able to exploit the fact that failed processes do not exhibit malicious behaviour to obtain performance

$$K \leq \left\lceil \frac{n-t}{t} \right\rceil^{-S}$$

This accords nicely with the results in [Fe], where it was found that for synchronous systems, the failure-by-omission model permits twice the rate of convergence allowed by the Byzantine failure model.

We also prove the matching lower bound

$$K \geq \left\lceil \frac{n-t}{t} \right\rceil^{-S}$$

for any deterministic t-resilient Approximate Agreement algorithm in an asynchronous system with failure-by-omission faults. This proof is the major original contibution of this paper, since the result is surprising at first, considering that in [Fe] it was found that for synchronous systems with failure-by-omission faults (or Byzantine faults) an S-round algorithm could do substantially better than an iterated round-by-round algorithm like those in [DLPSW] or this paper. The intuitive reason for this result is that the synchronous S-round algorithms of [Fe] exploit the fact that the same set of t processes have to account for the faulty behaviour in all the rounds. Thus the algorithms try to detect which processors are faulty, and then alter the information received from them to reduce the damage they can do. However in an asynchronous system with failure-by-omission, the worst damage a pro-

[1] Neglecting to send a message is included among the possible faults as a way of reconciling the tendency of real systems to lose messages from time to time with the failure-free network of our model.

[2] [DLPSW] demands that $w(p)$ lie in the interval of initial values of correct processors. Since we do not allow arbitrary maliciousness of faulty processors, there seems to be no reason to rule out their initial values.

[3] In the terminology of [MS], K is the ratio of final precision to initial precision.

cessor's failure can cause, is also produced by delaying its messages sufficiently.[4] Since delays do not have to involve the same processors in each round, there is no extra information to be obtained by carrying values over several rounds.

In §2 we introduce needed notation, and then we give a formal model of our system, and define precisely what it means for an algorithm to use only S rounds of communication in an asynchronous failure-by-omission system. Then we express formally the intuition above, that any results an "adversary" can obtain by causing processors to halt or not send messages, it can obtain without crashes or omissions, by delaying messages.[5] This result and the underlying intuition should have applications to other problems (e.g. Inexact Agreement [MS]) in asynchronous failure-by-omission systems. In §3 we give the algorithm and prove that it has the claimed performance. Then in §4 we prove the matching lower bound, using the result from §2. In §5 we discuss the fact that the algorithm and the lower bound are both also valid in an asynchronous crash-failure system, where faulty processors can only follow the algorithm or halt.

I would like to thank the members of the Theory of Distributed Systems group at M.I.T. for helpful discussions, and especially Jennifer Welch for eagle-eyed proofreading.

2 Notation and The System Model

We introduce the notion of a *multiset* of values. Formal definitions can be found in [DLPSW], but for us it will be enough to say that a multiset (sometimes called a bag) is an unordered collection of values which need not be distinct. For each value v and multiset V we denote the number of occurrences of v in V (the *multiplicity* of v) by $mult(v, V)$. The smallest interval containing all the values in V will be denoted by $\rho(V)$, and its length, the *diameter* of V, by $\delta(V)$, so $\rho(V) = [\min(V), \max(V)]$, and $\delta(V) = |\max(V) - \min(V)|$. We define the "average" that we will use by

$$\mathrm{av}(V) = \frac{v_1 + v_{t+1} + \cdots + v_{(\lambda-1)t+1}}{\lambda}$$

where $\lambda = \lceil \frac{N}{t} \rceil$ and the elements of the multiset V, in order from lowest to highest, are $v_1, v_2, \ldots, v_N$. In [DLPSW] the function av is called $f_{t,0}$. We use two lemmas which are special cases of Lemmas 4 and 5 from [DLPSW].

Lemma 1 *Suppose U and V are nonempty multisets with $V \subseteq U$. Then $\mathrm{av}(V) \in \rho(U)$.*

Lemma 2 *Suppose V, W, and U are nonempty multisets with $|V| = |W| = m$, $V \subseteq U$, $W \subseteq U$ and $|W - V| = |V - W| \leq t$. Then*

$$|\mathrm{av}(V) - \mathrm{av}(W)| \leq \frac{\delta(U)}{\lceil m/t \rceil}.$$

Now we give a formal model of an asynchronous failure-by-omission system, based closely on the most asynchronous model of [DDS], with asynchronous processors, communication, and message order, point-to-point transmission, and separate send and receive operations.[6] The main change to that model is that we allow multiple channels between each pair of processors, and allow a processor to try to receive messages from only a subset of channels, during a receive operation. This can be used to model the capacity in languages like CSP, for message receipt to be guarded by the message type. We also assume that there are initial and decision states for every real number, not just for 0 and 1 as in [DDS]. This formal model is not needed to discuss the algorithm or the lower bound, but it is needed for a rigorous proof of the intuition about the power of the adversary, mentioned in the introduction.

[4] This is reflected in [FLP] in the fact that the executions constructed there do not involve processors failing.

[5] This does not mean that an adversary without the power to cause crashes or omissions is as powerful as one with the power, but merely that (having the power) it need not exercise it!

[6] By results in [W], the processors could be assumed to be synchronous, without altering the results.

Formally, a protocol of the set P of processors $1, 2, \ldots, n$ is described by the following data: a universe M of messages, a collection C of channels for communication, and for each processor p a set of states Z^p and functions β^p (describing message generation), γ^p (describing the guards on message receipt), and δ^p (describing the state transitions). The collection of channels has two associated functions begin : $C \to P$ and end : $C \to P$. We put $C^{p,*} = \{c \in C : \text{begin}(c) = p\}$ and $C^{*,p} = \{c \in C : \text{end}(c) = p\}$, respectively the sets of channels starting and ending at p. We define the set E^p of events at p to be $\{\dagger, \emptyset\} \cup (C^{*,p} \times M)$, where $\dagger$ is a place holder representing a crashed processor's step, $\emptyset$ represents a step where no message is received at p, and (c, m) represents a step where message m is received by p from channel c. We form the set of events E as the tagged union of the set of events at each processor, $E = \cup_p \{p\} \times E^p$. We say that the event (p, e) involves processor p. The set Z^p is partitioned into a set of sending states Z^p_S and a set of receiving states Z^p_R. For each real number v there are distinguished an initial state with value v, $z^p_{v,init}$, and a set of decision states with value v, $Z^p_{v,dec}$. We require that each state be an initial or decision state for at most one value v. The message generation function $\beta^p : Z^p \to \wp(C^{p,*} \times M)$ is required to satisfy the conditions $|\beta^p(z)| \leq 1$ for all z, and $\beta^p(z) = \emptyset$ if $z \in Z^p_R$. Thus no message is generated during a receiving step, and at most one message is generated in a sending step. The guard function $\gamma^p : Z^p \to \wp(C^{*,p})$ is required to satisfy $\gamma^p(z) = \emptyset$ for $z \in Z^p_S$, modelling the fact that no messages are received during a sending step. The state transition function $\delta^p : Z^p \times E^p \to Z^p$ is required to satisfy the condition that $\delta^p(z, e) \in Z^p_{v,dec}$ for all $z \in Z^p_{v,dec}$ and for all e, to reflect the irrevocable nature of a decision. We also require that $\delta^p(z, \dagger) = z$ for all states z, since $\dagger$ reflects a place-holder for a step not taken because of a crash.

A configuration κ consists of a state for each processor and a multiset[7] of messages for each channel. We write $\text{st}(p, \kappa)$ specifying the state of processor p, and $\text{buff}(c, \kappa)$ for the messages in transit on channel c, in the configuration κ. We say the event (p, e) where $e \in E^p$ is applicable to the configuration κ, if either $e = \dagger$ or $e = \emptyset$ or $e = (c, m)$ where $c \in \gamma^p(\text{st}(p, \kappa))$ and $m \in \text{buff}(c, \kappa)$.

Suppose (p, e) is an event applicable to κ. If $e = (c, m)$ we define the failure-free result of (p, e) in κ to be the unique configuration κ' with $\text{st}(q, \kappa') = \text{st}(q, \kappa)$ for $q \neq p$, $\text{st}(p, \kappa') = \delta^p(\text{st}(p, \kappa), e)$ and $\text{buff}(d, \kappa') = \text{buff}(d, \kappa) \cup \{m' : (d, m') \in \beta^p(\text{st}(p, \kappa))\}$ for $d \neq c$ and $\text{buff}(c, \kappa') = (\text{buff}(c, \kappa) - \{m\}) \cup \{m' : (c, m') \in \beta^p(\text{st}(p, \kappa))\}$. If $e = \emptyset$ we define the failure-free result of (p, e) in κ to be the unique configuration κ' with $\text{st}(q, \kappa') = \text{st}(q, \kappa)$ for $q \neq p$, $\text{st}(p, \kappa') = \delta^p(\text{st}(p, \kappa), e)$ and $\text{buff}(d, \kappa') = \text{buff}(d, \kappa) \cup \{m' : (d, m') \in \beta^p(\text{st}(p, \kappa))\}$ for $d \in C$. If $e = (c, m)$ we also define the failure result of (p, e) in κ to be the unique configuration κ' with $\text{st}(q, \kappa') = \text{st}(q, \kappa)$ for $q \neq p$, $\text{st}(p, \kappa') = \delta^p(\text{st}(p, \kappa), e)$ and $\text{buff}(d, \kappa') = \text{buff}(d, \kappa)$ for $d \neq c$ and $\text{buff}(c, \kappa') = (\text{buff}(c, \kappa) - \{m\})$. If $e = \dagger$ or $e = \emptyset$ we define the failure result of (p, e) in κ to be the unique configuration κ' with $\text{st}(q, \kappa') = \text{st}(q, \kappa)$ for $q \neq p$, $\text{st}(p, \kappa') = \delta^p(\text{st}(p, \kappa), e)$ and $\text{buff}(d, \kappa') = \text{buff}(d, \kappa)$ for $d \in C$. Thus the failure result of an event (p, e) is produced by p altering its state as required by the transition function δ^p, but not adding a message to a buffer, even if required to do so by the message generation function γ.

A partial execution of the protocol is a finite, alternating sequence $\kappa_1, (p_1, e_1), \kappa_2, (p_2, e_2), \kappa_3, \ldots, (p_l, e_l), \kappa_{l+1}$ of configurations and events, starting and ending with a configuration, satisfying the conditions:

- for each processor p, $\text{st}(p, \kappa_1) = z^p_{v,init}$ for some v;
- for each channel c, $\text{buff}(c, \kappa_1) = \emptyset$;
- for each i, (p_i, e_i) is applicable to κ_i;
- for each i, κ_{i+1} is either the failure-free result or the failure result of (p_i, e_i) in κ_i;
- if $j > i$, $e_i = \dagger$, and $p_j = p_i$, then $e_j = \dagger$.

[7] A multiset or bag is used, rather than a set, because the same message could be sent several times.

An execution is an infinite alternating sequence $\kappa_1,(p_1,e_1),\kappa_2,(p_2,e_2),\kappa_3,\ldots$ whose prefixes of odd length are partial executions. In an execution or partial execution $\kappa_1,(p_1,e_1),\kappa_2,\ (p_2,e_2),\kappa_3,\ldots$, processor p is said to have failed if there is some i such that $p_i = p$ and κ_{i+1} is not the failure-free result of (p_i,e_i) in κ_i. Similarly, we say that processor p crashed in the execution or partial execution if there is some i such that $p_i = p$ and $e_i = \dagger$. A partial execution $\rho = \kappa_1,(p_1,e_1),\kappa_2,(p_2,e_2),\kappa_3,\ldots,(p_l,e_l),\kappa_{l+1}$ is called admissible if at most t processors have failed in ρ. We say that an execution $\rho = \kappa_1,(p_1,e_1),\kappa_2,(p_2,e_2),\kappa_3,\ldots$ is admissible if every prefix of odd length is an admissible partial execution and in addition, every processor is involved in an infinite number of the events (p_i,e_i), and also whenever $m \in \text{buff}(c,\kappa_i)$ then either there exists $j \geq i$ such that $e_j = (c,m)$, or else there are only a finite number of indices k such that $e_k \neq \emptyset$ and $c \in \gamma^{p_k}(\text{st}(p_k,\kappa_k))$. Thus in an admissible execution every processor takes steps (or has place-holders for failed steps) infinitely often, and every message sent is eventually received, if the addressee requests it often enough. We say an execution or partial execution ρ is failure-free if no processor has failed in ρ.

Given an execution or partial execution $\rho = \kappa_1,(p_1,e_1),\kappa_2,\ (p_2,e_2),\kappa_3,\ldots$, and a processor p, we say that p's initial state $\text{st}(p,\kappa_1)$, together with the subsequence of events (p_i,e_i) which involve p (i.e. for which $p_i = p$) form "the view of p in ρ". We note that $\text{st}(p,\kappa_{l+1})$ is uniquely determined by the view of p in the partial execution $\kappa_1,(p_1,e_1),\kappa_2,\ (p_2,e_2),\kappa_3,\ldots,(p_l,e_l),\kappa_{l+1}$.

We have the following useful extension result:

Lemma 3 *Let* $\rho = \kappa_1,(p_1,e_1),\kappa_2,(p_2,e_2),\kappa_3,\ldots,(p_l,e_l),\kappa_{l+1}$ *be an admissible, failure-free partial execution of a protocol. Then there is an admissible, failure-free execution* $\rho' = \kappa_1,(p_1,e_1),\kappa_2,(p_2,e_2),\kappa_3,\ldots$ *of the protocol which has* ρ *as a prefix.*

Proof: We inductively construct (p_i,e_i) and κ_{i+1} for $i > l$. Let $p_i = (i \bmod n) + 1$. Consider the multiset of messages $\{m : m \in \text{buff}(c,\kappa_i), c \in \gamma^{p_i}(\text{st}(p_i,\kappa_i))\}$, arranged in order, from the earliest sent to the latest sent. If the set is empty, let $e_i = \emptyset$, otherwise let $e_i = (c_0,m_0)$ where m_0 is the member of the multiset that was sent earliest, and where $m_0 \in \text{buff}(c_0,\kappa_i)$. Now, let κ_{i+1} be the failure-free result of (p_i,e_i) in κ_i.

By construction, ρ' is failure-free, and it is an execution. By the choice of p_i every processor takes an infinite number of steps in ρ' and by the choice of e_i every message is received if requested often enough. Thus ρ' is admissible. Q.E.D.

With this formal model, we prove the result (descibed in §1) about the behaviour of the "adversary" against which our algorithm needs to work.

Lemma 4 *Let* ρ *be an admissible execution of a protocol in an asynchronous failure-by-omission system, in which every processor that does not crash enters a decision state. Then there is an admissible, failure-free execution* $\tilde{\rho}$ *such that each processor that did not crash in* ρ *has the same view in* $\tilde{\rho}$ *as in* ρ*, up to the point where it enters a decision state (and thus it chooses the same decision value in both executions).*

Proof: Let $\rho = \kappa_1,(p_1,e_1),\kappa_2,(p_2,e_2),\kappa_3,\ldots$. Let R denote the set of processors that have not crashed in ρ. Choose some index l so that $\text{st}(p,\kappa_l)$ is a decision state for every processor $p \in R$. We will first construct an admissible, failure-free partial execution $\rho' = \kappa'_1,(p'_1,e'_1),\kappa'_2,(p'_2,e'_2),\ldots,\kappa'_i$, such that the view of p in ρ' is the same as the view of p in ρ. We put $\kappa'_1 = \kappa_1$. Now we define (p'_i,e'_i) and κ'_{i+1} inductively, for $i < l$. Let $p'_i = p_i$. If $e_i = \dagger$, let $e'_i = \emptyset$, and otherwise let $e'_i = e_i$. Let κ'_{i+1} be the failure-free result of (p'_i,e'_i) in κ'_i. We note that (p'_i,e'_i) is applicable to κ'_i since $\text{st}(p,\kappa'_i) = \text{st}(p,\kappa_i)$ as long as p has not crashed in the first i steps of ρ. Now we can use Lemma 3 to extend ρ' to an admissible, failure-free execution $\tilde{\rho}$. For $p \in R$, the view of p in ρ' is the same as its view in the prefix of ρ ending in κ_l, and (since therefore $\text{st}(p,\kappa'_l) = \text{st}(p,\kappa_l)$ is a decision state) we have thus completed the proof of the Lemma. Q.E.D.

A protocol in this formal model is said to solve the Approximate Agreement problem if in every admissible execution every processor that has not crashed eventually enters a decision state, and if the value of each decision state entered is within the range of the values of the initial states. The performance of such a protocol is the supremum (over all admissible executions) of the ratio of the diameter of the multiset of values of the decision states to the diameter of the multiset of values of the initial states. We say that such a protocol uses only S rounds of communication (or that it solves the S-round Approximate Agreement problem) if in every admissible execution every message sent has round number at most S, where we inductively specify the round number of a message as one higher than the round number of any message that the sender had received before sending the message in question, or one if the sender had not previously received any messages. In a completely synchronous system in which all processors exchange messages every round, this definition agrees with the round in which the message was sent. This definition explains why our model needs to include a mechanism for a protocol to be able to guard against receiving some messages too early — once a round r message has been received, a processor has lost the chance to send a round r message of its own.

3 The Algorithm

Each processor p acts according to the following algorithm, which we give first informally, as in [DLPSW]:

- Initially, assign val to be $v(p)$, the initial value of processor p.
- Next, for $r = 1, 2, \ldots, S$ successively
  - (i) Send a message $\langle r, val \rangle$ to each processor (including p itself).
  - (ii) Wait, trying to receive messages with r as the first component, and collecting the second components, until $n - t$ such messages have been received.
  - (iii) Arrange the $n - t$ values collected during step (ii) in increasing order. Select the lowest, $(t+1)$-st lowest, $(2t+1)$-st lowest, etc. Assign val to be the mean of the $\lceil \frac{n-t}{t} \rceil$ numbers selected.
- Finally decide on $w(p) = val$, and thereafter do not send or try to receive messages.

We refer to steps (i), (ii) and (iii) for r as forming "phase r".

Now we give a formal account of this protocol using the model of §2, to illustrate the correspondence between that model and the higher-level description above. The universe M of messages consists of all the real numbers. The channels are $c^{p,q,r}$ for p and q ranging from 1 to n, and r ranging from 1 to S, with $\text{begin}(c^{p,q,r}) = p$ and $\text{end}(c^{p,q,r}) = q$. We will use $c^{p,q,r}$ to carry the round r message from p to q. The state of each processor has components val (a real number), round (an integer), valrec (an array, indexed from 1 to n, with each value either a real number or $\perp$, indicating no value received so far), numbersent (an integer between 0 and $n-1$, inclusive), and mode (one of the values: send, receive, done). The receiving states Z_R^p are those where mode=receive or mode=done, and the sending states Z_S^p are those where mode=send. The initial state $z_{v,init}^p$ has val=v, round=1, valrec[i]=$\perp$ for all i, numbersent=0, and mode=send. The decision states $Z_{v,dec}^p$ are those with val=v and mode=done. The message generation function β^p is given by $\beta^p(z) = \{(c^{p,z.\text{numbersent}+1,z.\text{round}}, z.\text{val})\}$ if z.mode=send, and $\beta^p(z) = \emptyset$ otherwise. The message guard function is given by $\gamma^p(z) = \{c^{q,p,r} : r = z.\text{round}\}$ if z.mode=receive, and $\gamma^p(z) = \emptyset$ otherwise. The function δ^p is given by the following procedure, which takes as arguments a state z and an event e, and returns the value $\delta^p(z, e)$. The local variable w is initially set to z, and its components are then modified till its value is $\delta^p(z, e)$.

```
ComputeNextState(z:state,e:event):state
Local variable w:state
begin
w ← z
if e= †
  then return(w)
if z.mode=send
  then if z.numbersent< n − 1
         then begin
              w.numbersent ← w.numbersent+1
              return(w)
              end
         else begin
              w.numbersent ← 0
              w.mode=receive
              return(w)
              end
if z.mode=receive and e=∅
  then return(w)
if z.mode=receive and e=(c^{q,p,r}, m)
  then begin
       w.valrec[q]=m
       if |{i : w.valrec[i] ≠ ⊥}| < n − t
         then return(w)
         else begin
              w.val ← av({w.valrec[i] : w.valrec[i] ≠ ⊥})
              if w.round=S
                then begin
                     w.mode ←done
                     return(w)
                     end
                else begin
                     w.round ← w.round+1
                     for i = 1,...,n do w.valrec[i] ← ⊥
                     w.numbersent ← 0
                     w.mode ←send
                     return(w)
                     end
              end
       end
if z.mode=done
  then return(w)
end
```

The main properties of this algorithm all depend on the following lemma, which relates the values of the variables during successive rounds.

Lemma 5 *In an admissible execution of the protocol above, in which the initial value of processor p is $v(p)$, let $val_p^1 = v(p)$. Let val_p^{r+1} denote the value of val that is chosen in step (iii) of phase r. (Thus if p has not crashed, it sends messages $\langle r, val_p^r \rangle$ in step (ii) of phase r). Let U^r denote the multiset $\{val_p^r : p$ has not halted before executing step (iii) of phase $r\}$. Then*

$$\rho(U^{r+1}) \subseteq \rho(U^r)$$

and

$$\delta(U^{r+1}) \leq \left\lceil \frac{n-t}{t} \right\rceil^{-1} \delta(U^r)$$

Proof: It is clearly enough to prove that if p and q have not crashed before executing step (iii) of phase r, then

$$val_p^{r+1} \in \rho(U^r)$$

and

$$|val_p^{r+1} - val_q^{r+1}| \leq \left\lceil \frac{n-t}{t} \right\rceil^{-1} \delta(U^r)$$

Let V_p^r denote the multiset of $n-t$ round r values collected by p in step (ii) of phase r. We note that $V_p^r \subseteq U^r$, since processors cannot exhibit Byzantine behaviour, and so if a message $\langle r, v \rangle$ is received by p from p' then $v = val_{p'}^r$. Now $|V_p^r \cap V_q^r| + |V_p^r \cup V_q^r| = |V_p^r| + |V_q^r| = 2(n-t)$, but $V_p^r \cup V_q^r \subseteq U^r$, so $|V_p^r \cup V_q^r| \leq n$. Thus $|V_p^r \cap V_q^r| \geq n - 2t$, and so $|V_p^r - V_q^r| \leq t$. The conclusions now follow by Lemmas 1 and 2, since $val_p^{r+1} = \text{av}(V_p^r)$. Q.E.D.

Theorem 1 *The algorithm above solves the Approximate Agreement Problem using S rounds of communivation, with performance*

$$K \leq \left\lceil \frac{n-t}{t} \right\rceil^{-S}$$

Proof : Note that each message $\langle r, v \rangle$ is a round r message as defined in §2. The result is now immediate from the lemma. Q.E.D.

4 The Lower Bound

To simplify the analysis we will assume that any protocol considered has been put in a canonical form, as a *full-information* protocol[8]. A full-information protocol is the following:

- For $r = 1, 2, \ldots, S$ successively
  - (i) Send a message $\langle r, hist \rangle$ to each processor (including p itself) where *hist* is p's history (its view in the partial execution so far), that is, its initial value and a record of the messages that p received at each step.
  - (ii) Wait, trying to receive messages with 1,2,...,r as the first component, until there is a set of $n - t$ processors such that p has received r messages (one with each first component from 1 to r) from each of these processors during the execution.
- Finally decide on a value $w(p)$ which is some function of p's history, and thereafter do not send or try to receive messages.

Different full-information protocols are given by using different functions of the history to determine the decision value. We refer to steps (i) and (ii) for r as forming "phase r" of the execution.

We briefly explain the reasons this form of protocol is completely general, in that any protocol can be implemented by a full-information protocol. As we only consider deterministic algorithms, each message is determined by the history of the sender at the time the message is sent, so we can assume that it is the history itself that is sent. Since the receiver need not pay any attention to messages it is not interested in, there is no loss of generality in sending each message to every processor, or in sending a message for each round. Once a round r message is received, a processor cannot send a round r message of its own, so it should only try to receive messages from rounds up to $r - 1$ until it has sent its round r message. In order to put as much information as possible in that message, it should not send the round r message till it is possible that the processor has received everything that it will ever get, among the messages of rounds 1,...,$r - 1$. Thus, the processor should wait (trying to receive) as long as it knows that there are such messages still to come, but no longer. In the failure-by-omission asynchronous system, this means waiting until all messages of preceding rounds have been received from some set of $n - t$ processors, since the remaining t processors could have omitted to send all the messages not yet received from them.[9] Once a processor has entered a decision state, there is no point to trying to receive messages, as the final value in fixed, and any message sent after that point would violate the requirement that no message have round number greater than S. We also note that in an infinite admissible execution of a full-information protocol, every processor (unless it crashes) eventually decides on a final value.

To say that a protocol P has performance $K \geq \lceil \frac{n-t}{t} \rceil^{-S}$ is to say that there is some run (determined by some choice of initial values, processor failures and message delay times) in which two processors p and q reach decision states with values $w(p)$ and $w(q)$ such that $|w(p) - w(q)| \geq \lceil \frac{n-t}{t} \rceil^{-S} \delta$ where $\delta = \delta(v(1), \ldots, v(n))$ is the size of the interval of initial values in the run. We will prove a stronger statement, since the extra condition helps the induction argument work.

Theorem 2 *For any protocol P that solves the S-round t-resilient Approximate Agreement Problem in an asyn-*

[8] The full-information protocol we give here is a natural generalization to asynchronous systems of a standard form used for reasoning about synchronous algorithms.

[9] A detailed account of knowledge in failure-by-omission systems can be found in [MT].

chronous, failure-by-omission system, there is an admissible execution ρ (in which we denote the initial value of p' by $v(p')$) and processors p and q which enter decision states with final values $w(p)$ and $w(q)$ such that

$$|w(p) - w(q)| \geq \left\lceil \frac{n-t}{t} \right\rceil^{-S} \delta(v(1), \ldots, v(n))$$

and such that there are at most $\lceil \frac{n-2t}{t} \rceil t$ processors from which both p and q receive round S messages.

Proof: For notational convenience we put $\nu = \lceil \frac{n-t}{t} \rceil$. We first prove the theorem for $S = 1$.

We denote by p_α $(\alpha = 1, \ldots, \nu)$ the processor $\alpha t + 1$, and by $p_{\nu+1}$ processor 1. Now we will describe a chain of admissible executions ρ_0, ρ_1,...,$\rho_{\nu+1}$ such that processor p_α has the same history in executions $\rho_{\alpha-1}$ and ρ_α, and thus the same decision value in those executions. We will construct ρ_0 with each processor having initial value 1, so the decision value of p_1 in that execution must be 1. Similarly the decision value of $p_{\nu+1}$ in $\rho_{\nu+1}$ must be 0. From these facts it follows by a standard argument (see the lower bounds in [Fe] for example) that for some α the processes p_α and $p_{\alpha+1}$ reach decision states with final values that differ by at least ν^{-1}, in the admissible execution ρ_α which satisfies all the conditions of the theorem.

The execution ρ_0 is one where every processor has initial value 1, no processors crash or omit to send, and each processor receives round 1 messages from processes $t+1$,...,n before entering its decision state. For $\alpha = 1, \ldots, \nu - 1$ the execution ρ_α has processors 1,2,...,αt with initial value 0, and processors $\alpha t + 1$,...,n with initial value 1. No processor crashes or omits to send, and p_α enters its decision state after receiving round 1 messages from processors 1,...,$(\alpha - 1)t$ and $\alpha t + 1$,...,n, while every other processor (in particular $p_{\alpha+1}$) receives round 1 messages from processors 1,...,αt and $(\alpha + 1)t + 1$, ...,n before entering its decision state. The execution ρ_ν has processors 1,2,...,νt with initial value 0, and processors $\nu t + 1$,...,n with initial value 1. No processor crashes or omits to send, and p_ν enters its decision state after receiving round 1 messages from processors 1,...,$(\nu - 1)t$ and $\nu t + 1$,...,n, while every other processor (in particular $p_{\nu+1}$) receives round 1 messages from processors 1,...,$n - t$ before entering its decision state. In the execution $\rho_{\nu+1}$ every processor has initial value 0 and each processor enters its decision state after receiving messages from processors 1,...,$n - t$.

Now we suppose the theorem true for $(S-1)$-round protocols, and prove it for the S-round protocol $\mathcal{P}$.

From $\mathcal{P}$ we construct an $(S-1)$-round full-information protocol $\mathcal{Q}$. To describe $\mathcal{Q}$ we have to specify the decision value chosen by p after a given history h. This value will be the decision value chosen by p in protocol $\mathcal{P}$ after a history $\tilde{h}$. We now descibe $\tilde{h}$.[10] Let R_p denote a set of $n - t$ processors such that p received every message from each processor in R_p during h (such a set exists since p entered a decision state). The history $\tilde{h}$ is the same as h during stages 1,...,$S - 2$. During phase $S - 1$ of $\tilde{h}$, p receives only those messages (among those it received in h during phase $S - 1$) that were from processors in R_p. Then in phase S of $\tilde{h}$, p receives round S messages from the processors in R_p, and no other messages. We will have completed the description of $\tilde{h}$ when we give the contents of these round S messages. Since this is a full-information protocol, the message p receives from p' contains (beside the round number) a history of p'. This history is an extension of the history p received from p' in the round $S - 1$ message in h, with the extra events during phase $S - 1$ being the receipt by p' of any message (of rounds 1,...,$S-1$) from a processor in R_p that had not already been received by p'. We will see below that the history $\tilde{h}$ is actually a history of p that occurs in a run of protocol $\mathcal{P}$. Note that p can compute the history $\tilde{h}$, since it can compute the purported histories of processors p' (as

[10] We will give the description in the same high-level terminology we have used to descibe the protocols. For a complete description in the formal model of §2, we would also need to specify the order in which the messages are received during each phase. Any consistently applied choice would be suitable, for example, receiving a round i_1 message from processor p_1 before a round i_2 message from processor p_2 if $p_1 < p_2$ or ($p_1 = p_2$ and $i_1 < i_2$).

it did receive during h all the messages it is appending to the history of p').

The theorem applied to the $S-1$ round algorithm $\mathcal{Q}$ implies the existence of an admissible execution ρ' of $\mathcal{Q}$ and processors p' and q' which reach decision states with final values $w(p')$ and $w(q')$ satisfying

$$|w(p') - w(q')| \geq \left\lceil \frac{n-t}{t} \right\rceil^{-S+1} \delta(v(1), \ldots, v(n))$$

and there are at most $\lceil \frac{n-2t}{t} \rceil t$ processors from which both p' and q' receive round $S-1$ messages. Lemma 4 from §2 implies that we can assume that no processor is faulty during ρ'.

Choose processors $p_0, p_1, \ldots, p_\nu$ such that $p_0 = p'$, $p_\nu = q'$ and $p_\alpha \neq p_{\alpha+1}$ for $\alpha = 0, \ldots, \nu-1$. We will describe, in the next paragraph, admissible executions ρ_α for $\alpha = 1, \ldots, \nu$ of protocol $\mathcal{P}$ so that p_α has the same history in ρ_α and $\rho_{\alpha+1}$. Furthermore, the history of $p_0 = p'$ in execution ρ_1 is the same as the purported history constructed by p' during protocol $\mathcal{Q}$ in the execution ρ', and so during ρ_1, p_0 must decide on value $w(p')$. Similarly the history of $p_\nu = q'$ during execution ρ_ν will be the same as the history constructed by q' to determine its decision value during execution ρ' of protocol $\mathcal{Q}$, so during ρ_ν, p_ν must enter a decision state with value $w(q')$. Just as in the case $S = 1$, a standard argument shows that for some $\alpha = 1, \ldots, \nu$, the execution ρ_α causes processors $p_{\alpha-1}$ and p_α to enter decision states with final values $w(p_{\alpha-1})$ and $w(p_\alpha)$ such that

$$\begin{aligned} |w(p_{\alpha-1}) - w(p_\alpha)| &\geq \left\lceil \frac{n-t}{t} \right\rceil |w(p') - w(q')| \\ &\geq \left\lceil \frac{n-t}{t} \right\rceil^{-S} \delta(v(1), \ldots, v(n)) \end{aligned}$$

The construction of ρ_α also ensures that there are at most $(\nu-1)t$ processors from which $p_{\alpha-1}$ and p_α both receive round S messages during ρ_α, so that the admissible execution ρ_α satisfies all the conditions in the theorem.

Each execution ρ_α will be identical to ρ' for each processor during phases $1, \ldots, S-2$. Also, in each execution no processor will crash or omit to send any message. Thus we can describe each simply by indicating which messages are received by each processor during stages $S-1$ and S.[11] Let $M_{p'}(p)$ denote the set of messages (of rounds $1, \ldots, S-1$) sent from a processor in $R_{p'}$ to p during the execution ρ', except for those of these messages that had been received by p in stages $1, \ldots, S-2$. (Recall that $R_{p'}$ denotes the set of $n-t$ processors used in $\mathcal{Q}$ by processor p' to determine its decision value). Thus when, in the execution ρ' of protocol $\mathcal{Q}$, p' is constructing a history $\tilde{h}$, $M_{p'}(p)$ is the set of messages that processor p' appends to the history of p in the first $S-2$ rounds, to form a history that p' pretends it received in a round S message from p. Similarly, let $M_{q'}(p)$ denote the set of messages (of rounds $1, \ldots, S-1$) from a processor in $R_{q'}$ to p that had not been received by p in stages $1, \ldots, S-2$. Without loss of generality, we can renumber the processors so that the lowest numbered processors are those in both $R_{p'}$ and $R_{q'}$, and next come those in $R_{p'}$ but not $R_{q'}$, and then those in neither $R_{q'}$ nor $R_{p'}$, and finally those in $R_{q'}$ but not in $R_{p'}$. That is, since each of $R_{p'}$ and $R_{q'}$ contain $n-t$ processors, we can assume that $R_{p'} = \{1, \ldots, n-t\}$ and $R_{q'} = \{i : i \leq n-2t+\psi\} \cup \{i : n-t+\psi < i\}$, for some $\psi \geq 0$. The hypothesis of the induction is that $|R_{p'} \cap R_{q'}| \leq (\nu-1)t$, which implies that ψ satisfies $n-2t+\psi \leq (\nu-1)t$. Now we construct phase $S-1$ of the execution ρ_α for $\alpha = 1, \ldots, \nu$ by requiring that the set of messages received by processor p during phase $S-1$ be $M_{p'}(p)$ if $(\alpha-1)t < p \leq n-t$, and be $M_{q'}(p)$ otherwise.

The construction of phase S of the execution will be given separately for the cases $\nu \geq 2$ and $\nu = 1$. First suppose $\nu \geq 2$. In the execution ρ_1, during phase S, processor p_0 receives all the messages sent by processors $1, \ldots, n-t$ during phases $1, \ldots, S$ that it had not previ-

[11] In the formal model of §2, we would also need to specify in which order the processors take steps, and in which order the various messages in a phase are received, in order to completely specify the execution. For example, we could choose to let processors take steps in round robin order $1, \ldots, n, 1, 2, \ldots$, and similarly to let messages arrive in the order used for the formal description of protocol $\mathcal{Q}$.

ously received. In the execution ρ_1 every other processor (in particular p_1) receives in round S all the messages sent by processors $t+1,\ldots,n$ during phases $1,\ldots,S$, that it had not previously received. For $\alpha = 2,\ldots,\nu-1$, in the execution ρ_α the messages received by $p_{\alpha-1}$ are those from processors $1,\ldots,(\alpha-2)t$ and those from processors $(\alpha-1)t+1,\ldots,n$ (except for those of these messages that had been received before), while all processors except $p_{\alpha-1}$ receive the messages from processors $1,\ldots,(\alpha-1)t$ and from $\alpha t+1,\ldots,n$ that they had not received before. The execution ρ_ν has stage S where processor $p_{\nu-1}$ receives all outstanding messages from processors $1,\ldots,(\nu-2)t$ and from $(\nu-1)t+1,\ldots,n$, while all the processors except $p_{\nu-1}$ receive the outstanding messages from processors $1,\ldots,n-2t+\psi$ and from $n-t+\psi+1,\ldots,n$. In the case $\nu=1$ we need to construct only the execution ρ_1, with phase S in which processor p_0 receives all the as yet undelivered messages from processors $1,\ldots,n-t$ and each other processor receives all the messages from processors $t+1,\ldots,n$ that it had not received before.

It is now straightforward to see that the executions constructed above have the properties claimed for them, completing the proof of the induction step of the argument. Q.E.D.

5 The Crash-Failure Model

We now consider the approximate agreement problem in an asynchronous crash-failure system, which is just like the asynchronous failure-by-omission system discussed so far, except that the "adversary" is restricted in the ways it can have processors fail. In a crash-failure system processors may crash or they may operate correctly, but they can not omit to send a message and then continue functioning. In the formal model of §2, we say that an execution or partial execution is *crash-failure* if every configuration is the failure-free result of the previous step unless the previous step was $(p,\dagger)$ for some p. Since crash-failure executions form a subset of the failure-by-omission executions, it is obvious that any algorithm that solves the S-round approximate agreement problem in a failure-by-omission system will also solve the problem (with at least as good a performance) in a system where crashes are the only possible failures. However, it is (a priori) conceivable that there is some protocol that solves the problem in a crash-failure system, and that uses the special nature of the crash-failure system to obtain better performance than is possible for any algorithm in the more general failure-by-omission system. We show that this is not the case by converting any protocol for the crash-failure model into a general protocol, and then applying Theorem 2. Thus the lower bound of §4 also applies to the crash-failure model, and so the protocol of §3 remains optimal in the more restricted crash-failure system.

Theorem 3 *For any protocol $\mathcal{P}$ that solves the S-round t-resilient Approximate Agreement Problem in an asynchronous, crash-failure system, there is an admissible execution ρ' (in which the initial value of p' will be denoted $v(p')$) and processors p and q which enter decision states with final values $w(p)$ and $w(q)$ such that*

$$|w(p)-w(q)| \geq \left\lceil \frac{n-t}{t} \right\rceil^{-S} \delta(v(1),\ldots,v(n))$$

Proof: We first note that we may assume that $\mathcal{P}$ has the following form (which is a full-information protocol for the crash-failure model):

- For $r = 1,2,\ldots,S$ successively
  - (i) Send a message $(r, hist)$ to each processor (including p itself) where *hist* is p's history (its view in the partial execution so far), that is its initial value and a record of the messages that p received at each step.
  - (ii) Wait, trying to receive messages with 1,2,...,r as the first component, while p knows that some message remains in transit to p. That is, wait until p has received a round r_1 message from q (for every choice of r_1 and q such that

there is a round r_2 message from q with $r_2 > r_1$ among the messages p received, or among the messages whose receipt is reported in the histories that are the messages p received), and until there is a set of $n - t$ processors (including p itself) such that p has received r messages (one with each first component from 1 to r) from each of these processors during the execution.

- Finally decide on a value $w(p)$ which is some function of p's history, and thereafter do not send or try to receive messages.

We construct from $\mathcal{P}$ a protocol $\mathcal{Q}$ that is a full-information protocol for the failure-by-omission model as decribed in §4. Thus we need only specify the decision value chosen by processor p in $\mathcal{Q}$ after a history h, and this will be the decision value chosen by p in $\mathcal{P}$ after a history $\tilde{h} = \text{convert}^p_S(h)$, where convert^p_S is a function that we will define in the next paragraph, that converts an S-round history of p in the full-information failure-by-omission protocol to a history in the full-information crash-failure protocol.

We define inductively convert^p_r to convert a history h of p up to the end of phase r in the full-information failure-by-omission protocol, into a history of p up to the end of phase r in the full-information crash-failure protocol. If $r = 1$ then h consists of an initial value $v(p)$ for p, followed by receipt of $n - t$ messages containing initial values of processors. We define $\text{convert}^p_1(h)$ to be the history consisting of the same initial value $v(p)$ followed by receipt of the same messages as in h and also (unless h already contains the receipt of a round 1 message from p) by the receipt of a round 1 message from p itself, containing its own initial value. Now if $r > 1$, we let h' denote the prefix of h up to the end of phase $r - 1$ of the full-information protocol. We will form $\text{convert}^p_r(h)$ as an extension of $\text{convert}^p_{r-1}(h')$, with the additional events (forming phase r of the protocol) being receipt of certain messages described below. First, for any round r_1 message from p_1 (containing the history h_1 of p_1 up to the end of phase $r_1 - 1$) that p received during phase r in h, we include among the events of $\text{convert}^p_r(h)$ the receipt by p of a round r_1 message from p_1 containing the history $\text{convert}^{p_1}_{r_1-1}(h_1)$, except if $\text{convert}^p_{r-1}(h')$ already contains the receipt by p of a round r_1 message from p_1. Next, we include in $\text{convert}^p_r(h)$ the receipt by p of a round r message from itself containing the history $\text{convert}^p_{r-1}(h')$, unless a round r message from p is already among the messages whose receipt was added in the first step of the construction. Finally, we examine all the messages received by p in phase r of h or whose receipt is reported in a message received by p in phase r of h. For each such message, say a round r_2 message from p_2 containing a history h_2, and for each r_3 such that $r_3 < r_2$, we include in phase r of $\text{convert}^p_r(h)$ the receipt by p of a round r_3 message from p_2 containing the history $\text{convert}^{p_2}_{r_2-1}(h_3)$, where h_3 is the prefix of h_2 containing the events up to the end of phase $r_3 - 1$ of the full-information failure-by-omission protocol, unless the receipt by p of a round r_3 message from p_2 is among the events of $\text{convert}^p_{r-1}(h')$ or among the events included previosly into phase r of $\text{convert}^p_r(h)$. This completes the account of the computation of $\text{convert}^p_r(h)$, and therefore of the protocol $\mathcal{Q}$.

Let ρ be any admissible execution of the protocol $\mathcal{Q}$. By Lemma 4 there is an admissible failure-free execution ρ_1 of $\mathcal{Q}$ with the same initial values, and in which all processors that did not crash in ρ reach the same decision values. Then there is an admissible, crash-failure execution ρ'_1 of $\mathcal{P}$, such that for each processor p the view of p in ρ'_1 is $\text{convert}^p_S(h)$, where h is the view of p in ρ_1. Thus the initial values and (by the construction of $\mathcal{Q}$) the decision values chosen in ρ_1 are just the same as those in ρ'_1. Since $\mathcal{P}$ solves the Approximate Agreement problem we know that the decision values chosen in ρ'_1 are within the range of the initial values. Therefore the decision values chosen in ρ_1 (and hence in ρ) are within the range of

the initial values, and so we see that $\mathcal{Q}$ solves the Approximate Agreement problem in the failure-by-omission system. Thus from Theorem 2 from §4, we deduce the existence of a particular execution ρ of $\mathcal{Q}$, for which there are processors p and q whose decision values satisfy

$$|w(p) - w(q)| \geq \left\lceil \frac{n-t}{t} \right\rceil^{-s} \delta(v(1), \ldots, v(n))$$

The corresponding execution ρ_1' of $\mathcal{P}$ therefore has the properties required for this theorem. Q.E.D.

[DDS] D. Dolev, C. Dwork, L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus", JACM, to appear.

[DLPSW] D. Dolev, N. Lynch, S. Pinter, E. Stark, W. Weihl, "Reaching Approximate Agreement in the Presence of Faults", JACM, 33, 3, 499–516 (1986).

[Fe] A. Fekete, "Asymptotically Optimal Algorithms for Approximate Agreement", Proceedings of the 5th ACM Symposium on Principles of Distributed Computing, 73–87, August 1986.

[Fi] M. Fischer, "The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)" Yale University Technical Report YALEU/DCS/RR-273 (1983).

[FLP] M. Fischer, N. Lynch, M. Patterson, "Impossibility of Distributed Consensus with One Faulty Process", JACM, 32, 2, 374–382 (1985).

[HSSD] J. Halpern, B. Simons, R. Strong, D. Dolev, "Fault-tolerant Clock Synchronization", Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, 89–102, August 1984.

[LL] J. Lundelius, N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization", Information and Control, 62, 2, 190–204 (1984).

[LM] L. Lamport, P. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults", JACM, 32, 1, 52–78 (1985).

[MS] S. Mahaney, F. Schneider, "Inexact Agreement: Accuracy, Precision and Graceful Degradation", Proceedings of the 4th ACM Symposium on Principles of Distributed Computing, 237–249, August 1985.

[MT] Y. Moses, M. Tuttle, "Programming Simultaneous Actions Using Common Knowledge", Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 208–221, October 1986.

[PSL] M. Pease, R. Shostak, L. Lamport, "Reaching Agreement in the Presence of Faults", JACM 27, 2, 228–234 (1980).

[W] J. L. Welch, "Simulating Synchronous Processors", Information and Control, to appear.