Approximate Agreement

Alan D. Fekete
Laboratory for Computer Science
Massachusetts Institute of Technology

Abstract: This paper studies the problem of Approximate Agreement. This is a generalization of Byzantine Agreement, where processors are only required to obtain values that are close together, rather than identical. We offer new multi-round algorithms in several different models of computation, distinguished by the degree of synchrony in the system and the malevolence allowed to faulty processors. For each model we also examine the theoretical limits on attainable performance (measured by the reduction in the range of values), and show that our algorithm is asymptotically optimal with increasing ratio of non-faulty processors to faulty ones. There are two main conclusions we can draw from these algorithms and lower bounds. First, if process failures are restricted to faults of omission only (that is, a faulty processor is not allowed to send a wrong message, although it is allowed to crash, and therefore not send any message) then twice as much reduction can be achieved in each round of the algorithm as in a model where faults of commission are possible. This relationship holds in both synchronous and asynchronous systems. Second, we show that in synchronous systems algorithms that combine information from different rounds of message exchange can perform better than algorithms that treat each round separately. This extra performance is obtained by detecting which processors are faulty, and removing them from the system. In contrast, in asynchronous systems with faults of omission only there is no way to improve performance by using multiple rounds together rather than independently.

Keywords: Approximate Agreement, failure models, consensus protocols.

September 1987

Approximate Agreement[1]

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]. This problem 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). In synchronous systems (where processors operate in a sequence of rounds) it has been found that $t+1$ rounds of communication are needed in the worst case for algorithms that are resilient to t faulty processors ([FL]). Even worse is 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, as 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 ([LaM], [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]. In that paper algorithms were given for both synchronous and

[1]This paper forms part of the author's Ph.D. thesis "Topics in Distributed Algorithms", Department of Mathematics, Harvard University, August 1987. A preliminary version of the material in §§2–8 of this paper has appeared as "Asymptotically Optimal Algorithms for Approximate Agreement" in the Proceedings of the 5th ACM Symposium on Principles of Distributed Computing (August 1986). A preliminary version of §§9–12 has appeared as "Asynchronous Approximate Agreement" in the Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (August 1987). The work was supported in part by the Office of Naval Research under Contract N00014-85-K-0168, by the Office of Army Research under contract DAAG29-84-K-0058, by the National Science Foundation under Grants MCS-8306854, DCR-83-02391, and CCR-8611442, and by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125.

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. We provide algorithms that combine information from all rounds, and by this means have faster rates of convergence, in three different synchronous systems (Byzantine failure, failure-by-omission and crash-failure). We also prove lower bounds on the achievable rate of convergence, to show that each of the algorithms we give has performance that is asymptotically (as the number of processors increases) optimal. In contrast, we give an algorithm using independent rounds, and show that it is optimal, for asynchronous systems in which processor failures are relatively benign (failure-by-omission and crash-failure).

The Approximate Agreement Problem is studied in the following form: there are n processors labelled $1,2,\ldots,n$ that are linked by a completely connected, fault-free, point-to-point network that is the only means of interprocess communication. In synchronous models processors all take a step at once, and any message sent in such a step will arrive at the destination by the next step. In asynchronous models, 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. In each execution there is some subset $Corr$ of processors (the correct ones), so that if $p \in Corr$ then p executes the given algorithm. Independent of the degree of synchrony, we can consider three models of computation distinguished by the flexibility of behavior of the other (faulty) processors. In the *crash-failure* model a faulty processor executes the given protocol up to some point and then halts (without loss of generality we assume the crash doesn't occur in the middle of sending a message). In the *failure-by-omission* model a faulty processor may neglect to send a message that the protocol calls for it to send, and it may halt, but it does not send any message that is different from what the protocol requires. The most general model is the *Byzantine* model, in which a faulty processor may change state or send a message arbitrarily. We denote the set of faulty processors by $\mathit{Fault} = \{1, 2, \ldots, n\} \setminus \mathit{Corr}$ and set $f = |\mathit{Fault}|$. We also denote the subset of Fault, consisting of those processors which halt ("crash") during the execution, by Crash. Each processor p has an initial value $v(p)$ which is a real number. In the crash-failure and failure-by-omission models, we require that after any execution of the algorithm for which $f \leq t$, each processor p that has not crashed must arrive at a new value $w(p)$ satisfying a validity condition: that $w(p)$ must lie within the range of the initial values. In the Byzantine model we do not trust the initial values of faulty processors and we do not make demands of faulty processors' final state, so we insist that after any execution for which $f \leq t$ each correct p must arrive at a final value $w(p)$ that must lie within the range of the initial values of the correct processors. In none of the models do we put any requirement on the behavior of correct processors when more than t processors are faulty.

We denote the smallest interval containing a collection of values V by $\rho(V)$ and its length, the diameter of V, by $\delta(V)$ so that $\rho(V)$ is the interval $[\min(V), \max(V)]$ and $\delta(V) = \max(V) - \min(V)$. Let us denote by U the collection of initial values of all processors and by $\hat{U}$ the collection of initial values of correct processors, so $U = \{v(p)\}$ and $\hat{U} = \{v(p) : p \in \mathit{Corr}\}$. We can express the validity condition in the failure-by-omission and crash-failure models by "if $|\mathit{Fault}| \leq t$ and $p \notin \mathit{Crash}$ then $w(p) \in \rho(U)$". Similarly in the Byzantine model the validity condition is "if $|\mathit{Fault}| \leq t$ and $p \in \mathit{Corr}$ then $w(p) \in \rho\left(\hat{U}\right)$".

We will measure the performance of such an algorithm by the change in the range spanned by the values of the processors. Thus we measure performance in the crash-failure and failure-by-omission models by $K = \sup \dfrac{\delta(\{w(p) : p \notin \mathit{Crash}\})}{\delta(U)}$ and in the Byzantine model by $K = \sup \dfrac{\delta(\{w(p) : p \in \mathit{Corr}\})}{\delta\left(\hat{U}\right)}$, in each case the supremum being taken over all executions with $|\mathit{Fault}| \leq t$ (so a good algorithm is one with a low value for K). Notice that the identification of processors as faulty or correct is not known to the processors during the algorithm.

The results of [DLPSW] indicate that in both synchronous and asynchronous Byzantine systems, with n large enough, any value for K can be achieved if enough communication is used, so we will restrict our discussion to algorithms using at most S rounds of communication.

For the synchronous, Byzantine failure model, the paper [DLPSW] gives an algorithm using only one round of communication, valid when $n > 3t$, with performance $K = \lceil (n-2t)/t \rceil^{-1}$. In [DLPSW] it is shown that this is optimal if only one round of communication is allowed. We can clearly iterate this algorithm (that is, use the final values produced by one execution as initial values in another and then use the final values of that as initial values in a third execution, and so on for S rounds). This gives an S-round solution with $K = (\lceil (n-2t)/t \rceil)^{-S}$. We introduce for this model an S-round algorithm valid when $n > 4t$, with performance

$$K \leq \frac{\sup\left(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ each } l_i \text{ a nonnegative integer}\right)}{(n-2t)\,(n-4t)^{S-1}}.$$

By elementary calculus this supremum is at most t^S/S^S so we see

$$K \leq \frac{t^S}{S^S\,(n-2t)\,(n-4t)^{S-1}}$$

which for large n is asymptotic to S^S times better than the performance of the iterated algorithm of [DLPSW]. In fact as $n/t \to \infty$ so the number of processors increases relative to the number faulty, this performance is asymptotic to the best possible for a synchronous S-round algorithm resilient to t Byzantine failures by the lower bound

$$K \geq \frac{\sup\left(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ each } l_i \text{ a nonnegative integer}\right)}{(n+t)^S}$$

whose asymptotic form (and proof sketch) were first given in [DLPSW]. An interesting feature of our algorithm is that each processor tries to identify which of the other processors is faulty, and then ignores any information received from a known faulty processor to reduce the possibilities for disagreement. The technique of detection of faulty processors was first used in the similar problem of inexact agreement (where there is an a priori bound on the spread of initial values of correct processors) in [MS].

We give a new lower bound for K in the synchronous crash-failure model, namely

$$K \geq \frac{\sup\left(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ each } l_i \text{ a nonnegative integer}\right)}{(2n+3t)^S}$$

for any algorithm using S rounds of communication. We also give an algorithm for the crash-failure model, valid when $n > t$, with performance

$$K \leq \frac{\sup\left(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ each } l_i \text{ a nonnegative integer}\right)}{(2n-2t)^S}$$

which is asymptotic to the optimum as n/t increases.

We offer an algorithm in the synchronous failure-by-omission model, valid when $n > 2t$, by combining parts of the algorithms from the other synchronous models. This has performance

$$K \leq \frac{\sup\left(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ each } l_i \text{ a nonnegative integer}\right)}{(2n-4t)^{S-1}\,(2n-2t)}$$

which is again asymptotic to optimal.

It is worth noting that if $S = t+1$ the expression $\sup(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t$, each l_i a nonnegative integer) is zero, as one of the l_i must be zero, and so our synchronous algorithms give solutions to the exact agreement problem when run for $t+1$ rounds. In the Byzantine model this solution satisfies the strong validity condition that the value agreed on lies in the range of initial values of correct processors. (This is not achieved by using a normal Byzantine agreement algorithm on each bit of the initial values, unless some removal of extreme values is done.) In each model our algorithm for S rounds starts by doing all the communication of the $S-1$ round algorithm, so it is possible to do approximate agreement without knowing at the start how many rounds will be used. In fact, after each round the new values can be calculated as if that round were the last – this permits the values held by the correct processors to approach one another rapidly, finally agreeing if $t+1$ rounds are used.

For the asynchronous Byzantine failure model, [DLPSW] gives an iterated algorithm with performance

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

For the asynchronous crash-failure model we give an iterated algorithm, valid when $n > t$, which is very similar to that of [DLPSW], but 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 the synchronous case, where we found that the failure-by-omission model permits twice the rate of convergence per round 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 S-round t-resilient Approximate Agreement algorithm in an asynchronous system with failure-by-omission faults, and then extend this to apply also to an asynchronous crash-failure system. This result is surprising at first, considering that in synchronous systems we saw above that an S-round algorithm could do substantially better than an iterated round-by-round algorithm. The intuitive reason for this result is that the synchronous S-round algorithms 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 faults, the worst damage a processors' failure can cause is also produced by delaying its messages sufficiently. 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 intoduce the notation we will use, and prove the preliminary lemmas concerning the operators for removing extreme values. In §3, we discuss a simple algorithm that serves to introduce the main ideas for all the algorithms in the synchronous models. In §4 we give the asymptotically optimal algorithm for the synchronous Byzantine failure model, and in §5 we prove the lower bound for this model. Next, in §6 and §7, we discuss the synchronous crash-failure model, and then in §8, we combine ideas from the previous algorithms to produce an algorithm for the synchronous failure-by-omission system. In §9 we begin our discussion of asynchronous systems, with a careful formal model of an asynchronous failure-by-omission system. We use this model to prove a lemma that captures the intuition about the damage caused by processor failures being achievable with only message delays. Then in §10 we give the algorithm for the asynchronous failure-by-omission system, and in §11 we prove the matching lower bound. In §12 we extend the discussion to an asynchronous crash-failure model. Finally in §13 we summarize the results of this paper.

2 Notation and Lemmas

In order to give the algorithms precisely, we introduce the language of multisets. A formal account appears in [DLPSW] but for our purposes it is enough to think of a multiset as an unordered collection of values that 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 values may be either real numbers or the special symbols $\perp_r$ denoting a value not received in round r because (for example) a processor failed to send it. We define union, intersection, cardinality, max, min, and mean for multisets in the obvious ways, eg for any v, $mult(v, V \cap W) = min(mult(v, V), mult(v, W))$, $mult(v, V \cup W) = mult(v, V) + mult(v, W)$, and $|V| = \sum_v mult(v, V)$. Also let $double(V)$ be defined by $mult(v, double(V)) = 2 \cdot mult(v, V)$.

As in [DLPSW] we will try to reduce the range of values held by processors by using operators that act on multisets by removing extreme values. Let V be a multiset with $|V| = N$. We put $red_k(V)$ to be the multiset with $N - 2k$ entries formed from V by removing the k highest entries and also the k lowest entries. We order the values by treating $\perp_r$ as greater than any real number and also as greater than $\perp_R$ if $r > R$. For the crash-failure or failure-by-omission models we will use similar operators $chop_k^r$ that prefer to remove as many occurrences as possible of $\perp_r$, rather than removing other values. If $|V| = N$ and $mult(\perp_r, V) = j$ then $chop_k^r(V)$ is a multiset of $2N - 2k$ entries formed from $double(V)$ either by removing $2k$ copies of $\perp_r$ (in the case $j > k$) or else by removing all $2j$ copies of $\perp_r$ and then removing the $k - j$ highest and $k - j$ lowest of the remaining entries.

We similarly have operators to find a single number to be an "average" for a multiset. Suppose $|V| = N$ and at least $N - k$ entries in V are real numbers. Then we put $mid_k(V) = mean(red_k(V))$. Similarly if $|V| = N$, at least $N - k$ entries of V are real numbers and $mult(\perp_r, V) = 0$ for $r > 1$ we define $center_k(V) = mean(chop_k^1(V))$. The facts below and the conditions given in each case will ensure that, in our algorithms, a mean is only taken for multisets of real values. In asynchronous systems, we will use another averaging function, defined when all the elements of V are real by

$$av_k(V) = \frac{v_1 + v_{k+1} + \cdots + v_{(\lambda-1)k+1}}{\lambda}$$

where $\lambda = \lceil \frac{N}{k} \rceil$ and the elements of the multiset V, in order from lowest to highest, are

$v_1, v_2, \ldots, v_N$. Thus this function selects every k-th entry of the multiset and then takes the mean of these. In [DLPSW] the function av_k is called $f_{k,0}$. As examples:

- $\{-1,-1,0\} \cup \{0,1\} = \{-1,-1,0,0,1\}$
- $\{-1,-1,\bot_1\} \cup \{0,\bot_1,\bot_2\} = \{-1,-1,0,\bot_1,\bot_1,\bot_2\}$
- $\{-1,-1,0,0\} \cap \{-1,0,0,1\} = \{-1,0,0\}$
- $|\{-1,-1,0\}| = 3$
- $|\{-1,-1,0,\bot_1\}| = 4$
- $red_2(\{-1,-1,-1,0,0,1\}) = \{-1,0\}$
- $red_1(\{-1,-1,0,\bot_1,\bot_2\}) = \{-1,0,\bot_1\}$
- $chop_1^2(\{-1,0,0,\bot_2,\bot_2\}) = \{-1,-1,0,0,0,0,\bot_2,\bot_2\}$
- $chop_3^2(\{-1,0,0,\bot_2,\bot_2\}) = \{-1,0,0,0\}$
- $mid_2(\{-1,-1,-1,0,1,\bot_1\}) = -0.5$
- $center_3(\{-1,-1,0,1,\bot_1\}) = -0.5$
- $center_2(\{-1,-1,0,1,\bot_1\}) = -1/3$
- $av_2(\{-1,-1,0,2,5\}) = 4/3$

In our discussion we will need to know how the operators introduced affect the range of values in a multiset and the differences between two multisets. We have the following results:

Lemma 1 [DLPSW] *If V is a multiset with $|V| = N$, and at least $N-k$ elements of V lie in the range $[a,b]$, then every element of $red_k(V)$ lies in the range $[a,b]$.*

Proof: At most k elements of V are greater than b and all of these must be removed among the k highest elements of V when forming $red_k(V)$. Thus every element of $red_k(V)$ is less than or equal to b, and a symmetric argument shows that every element of $red_k(V)$ is greater than or equal to a. Q.E.D.

Lemma 2 [DLPSW] *If V and W are multisets then $|red_k(V) \cap red_k(W)| \geq |V \cap W| - 2k$.*

Proof: Since $V \cap W \subseteq V$, $red_k(V \cap W) \subseteq red_k(V)$ and similarly $red_k(V \cap W) \subseteq red_k(W)$, so $red_k(V \cap W) \subseteq red_k(V) \cap red_k(W)$, but $|red_k(V \cap W)| = |V \cap W| - 2k$. Q.E.D.

Lemma 3 *If V is a multiset with $|V| = N$, $mult(\perp_r, V) \leq k$ such that at least $N - k$ entries of V are different from $\perp_r$ and lie in the interval $[a, b]$, then every entry of $chop^r_{2k}(V)$ lies in $[a, b]$.*

Proof: Let $mult(\perp_r, V) = j$ and let Z denote the multiset of $2N - 2j$ entries formed from $double(V)$ by removing all $2j$ copies of $\perp_r$. Now $chop^r_{2k}(V) = red_{2k-j}(Z)$, and at least $2N - 2k$ entries of Z lie in $[a, b]$ so Lemma 1 completes the proof. Q.E.D.

Lemma 4 *Let V and W be multisets with $|V| = |W| = N$. Suppose that every entry in $V \cup W$ is one of v, w or $\perp_r$, and that $mult(\perp_r, V) \leq k$ and $mult(\perp_r, W) \leq k$. If $|mult(v, V) - mult(v, W)| + |mult(w, V) - mult(w, W)| \leq m$ then $|mult(v, chop^r_k(V)) - mult(v, chop^r_k(W))| \leq m$ and $|mult(w, chop^r_k(V)) - mult(w, chop^r_k(W))| \leq m$.*

Proof: Without loss of generality we may assume $v < w$. We first observe that W can be formed from V by a sequence of at most m operations, each being the replacement of a single entry by $\perp_r$ or the replacement of a single occurrence of $\perp_r$ by either v or w. Thus it is enough to prove that $|mult(v, chop^r_k(V_1)) - mult(v, chop^r_k(V_2))| \leq 1$ when $mult(\perp_r, V_1) \leq k - 1$ and V_2 is formed from V_1 by removing a single occurrence of z (which is either v or w) and replacing it with $\perp_r$. So we put $j = mult(\perp_r, V_1)$ and let Z denote the multiset of $2N - 2j$ entries formed by removing all occurrences of $\perp_r$ from $double(V_1)$. Now $chop^r_k(V_1)$ is formed from Z by removing the $k - j$ highest entries and the $k - j$ lowest entries. On the other hand, $chop^r_k(V_2)$ is formed from Z by removing two occurrences of z and then removing the $k - j - 1$ highest and $k - j - 1$ lowest of the remaining entries. If $z = v$ this is equivalent to removing the $k - j - 1$ highest and $k - j + 1$ lowest entries from Z as v is the lowest entry in Z, while if $z = w$ the net effect is to remove the $k - j + 1$ highest and $k - j - 1$ lowest entries from Z. Thus we can obtain $chop^r_k(V_2)$ from $chop^r_k(V_1)$ either by removing an occurrence of the $k - j + 1$ lowest entry of Z and adding an occurrence of the $k - j$ highest entry of Z, or else by replacing an occurrence of the $k - j + 1$ highest entry of Z by the $k - j$ lowest entry of Z. In either case we see that the multiplicities of v and w can each change by at most 1.

Q.E.D.

Lemma 5 *Suppose V and W are multisets with $|V| = |W| = N$, $|V \cap W| \geq N - m$ and at least $N - k$ elements of each of V and W lie in the interval $[a, b]$. Then $mid_k(V)$ and $mid_k(W)$ lie in $[a, b]$ and $|mid_k(V) - mid_k(W)| \leq m(b - a)/(N - 2k)$.*

Proof: By Lemma 1 we see that all the entries of $red_k(V)$ lie in the interval $[a, b]$ and so their average $mid_k(V)$ also lies in $[a, b]$. Similarly every entry of $red_k(W)$ and also $mid_k(W)$ lie in $[a, b]$. By Lemma 2, the multisets $red_k(V)$ and $red_k(W)$ agree in at least $N - 2k - m$ of their entries, and in each of the remaining m places, the entries can differ by at most $b - a$ as each lies in $[a, b]$. Thus $|mid_k(V) - mid_k(W)| = \frac{1}{N-2k}|\sum red_k(V) - \sum red_k(W)| \leq m(b - a)/(N - 2k)$.

Q.E.D.

Lemma 6 *Suppose V and W are multisets with $|V| = |W| = N$, such that $mult(\perp_1, V) \leq k$, $mult(\perp_1, W) \leq k$, $mult(\perp_r, V) = mult(\perp_r, W) = 0$ for $r > 1$, all real entries of $V \cup W$ lie in the interval $[a, b]$ and $\sum_{v \neq \perp_1} |mult(v, V) - mult(v, W)| \leq m$. Then $center_k(V)$ and $center_k(W)$ lie in $[a, b]$ and $|center_k(V) - center_k(W)| \leq m(b - a)/(2N - 2k)$.*

Proof: The hypotheses show that in $double(V)$ there will be at most $2k$ entries that are not real, and all of them will be $\perp_1$ and so will be removed in forming $chop_k^1(V)$. Thus the resulting multiset has all its entries in $[a, b]$ and so its mean $center_k(V)$ also lies in $[a, b]$. Similarly $center_k(W)$ also lies in $[a, b]$. Now as in the proof of Lemma 4 we observe that W is formed from V by at most m operations each replacing a value by $\perp_1$ or vice versa. So we need only prove that if V_1 and V_2 are multisets with $|V_1| = N$, $mult(\perp_1, V_1) \leq k - 1$, $mult(\perp_r, V_1) = 0$ for $r > 0$, and every real entry of V_1 lies in the interval $[a, b]$ and such that V_2 is formed from V_1 by removing one occurrence of a value z and replacing it with $\perp_1$, then $|center_k(V_1) - center_k(V_2)| \leq (b - a)/(2N - 2k)$. So we put $j = mult(\perp_1, V_1)$ and let Z denote the multiset of $2N - 2j$ entries formed by removing all occurrences of $\perp_1$ from $double(V_1)$. Now $chop_k^1(V_1)$ is formed from Z by removing the $k - j$ highest entries and the $k - j$ lowest entries. On the other hand, $chop_k^1(V_2)$ is formed from Z by removing two occurrences of z and then removing the $k-j-1$ highest and $k-j-1$ lowest of the remaining entries. If z is among the $k-j-1$ lowest entries of Z, this is equivalent to removing the $k-j-1$ highest and $k-j+1$ lowest entries from Z. If z is among the $k-j-1$ highest entries of Z the net effect is to remove the $k-j+1$ highest and $k-j-1$ lowest entries from Z. Thus in these cases, we can obtain $chop^1_k(V_2)$ from $chop^1_k(V_1)$ either by removing an occurrence of the $k-j+1$ lowest entry of Z and adding an occurrence of the $k-j$ highest entry of Z, or else by replacing an occurrence of the $k-j+1$ highest entry of Z by the $k-j$ lowest entry of Z. Clearly in these cases, the sum of the entries of $chop^1_k(V_1)$ differs from the sum of the entries of $chop^1_k(V_2)$ by the difference of two elements of the interval $[a,b]$ which is at most $b-a$. In the remaining case z lies between the $k-j$ lowest entry of Z (call it a') and the $k-j$ highest entry of Z (call it b'), but $chop^1_k(V_2)$ is obtained from $chop^1_k(V_1)$ by removing two occurrences of z and replacing them with a' and b' which will alter the sum of the entries by $b'+a'-2z$ which is at most $b'-z$ (as $z \geq a'$) but this is bounded by $b-a$. Thus in every case

$$|center_k(V_1) - center_k(V_2)| = \frac{1}{2N-2k}|\sum chop^1_k(V_1) - \sum chop^1_k(V_2)| \leq \frac{b-a}{2N-2k}$$

as required. Q.E.D.

Lemma 7 [DLPSW] *Suppose U and V are nonempty multisets with $V \subseteq U$. Then $av_k(V) \in \rho(U)$.*

Proof: Since every element of V lies within $\rho(U)$, so must the mean of a collection of such values. Q.E.D.

Lemma 8 [DLPSW] *Suppose V, W, and U are nonempty multisets with $|V| = |W| = m$, $V \subseteq U$, $W \subseteq U$ and $|W - V| = |V - W| \leq k$. Then*

$$|av_k(V) - av_k(W)| \leq \frac{\delta(U)}{\lceil m/k \rceil}.$$

Proof: Let the elements of V, arranged in increasing order, be $v_1, v_2, \ldots, v_m$. Similarly let the elements of W in increasing order be $w_1, \ldots, w_m$. Since $|W - V| \leq k$, we can deduce $v_i \leq w_{i+k}$, for any i such that $i \leq m-k$. Similarly $w_i \leq v_{i+k}$. Now,

$$av_k(V) - av_k(W) \;=\; \frac{1}{\lambda} \cdot ((v_1 + v_{k+1} + \ldots + v_{(\lambda-1)k+1}) - (w_1 + w_{k+1} + \ldots + w_{(\lambda-1)k+1}))$$

$$\begin{aligned}
&\le \frac{1}{\lambda}\cdot((w_{k+1}+w_{2k+1}+\ldots+w_{(\lambda-1)k+1}+v_{(\lambda-1)k+1}) \\
&\qquad -(w_1+w_{k+1}+\ldots+w_{(\lambda-1)k+1})) \\
&= \frac{v_{(\lambda-1)k+1}-v_1}{\lambda} \\
&\le \frac{\delta(U)}{\lceil m/k\rceil}.
\end{aligned}$$

Similarly we have that $av_k(W)-av_k(V)\le\frac{\delta(U)}{\lceil m/k\rceil}$. Q.E.D.

3 Introduction to the Algorithms for Synchronous Systems

The algorithms given for synchronous systems are all variants on a single plan. To help the reader understand them we give here an account of a basic algorithm for the crash-failure model. This algorithm is not optimal, but it is simpler than the others while still capturing the essential features, and it will isolate the main issues involved in solving the approximate agreement problem. For ease of exposition in this and the later algorithms, we will suppose that when a processor broadcasts information it sends to itself as well as to the other processors, though in practice this will usually be implemented by remembering, rather than sending a message.

In the basic algorithm, processor p, until it crashes, must perform the following –

- In round 1: Broadcast $v(p)$, and denote by $v(q_1,p)$ the value received by p from q_1 which was $v(q_1)$. If the message from q_1 is missing set $v(q_1,p)$ to be $\perp_1$.
- In round r, for $r=2,\ldots,S$, processor p will start with an array of n^{r-1} values $\langle v(q_1,q_2,\ldots,q_{r-1},p)\ :\ \text{each } q_i = 1,\ldots,n\rangle$. Now p should broadcast the array $\langle v(q_1,q_2,\ldots,q_{r-1},p)\rangle$. Denote by $v(q_1,\ldots,q_{r-1},q_r,p)$ the value received by p from q_r which was held by q_r as $v(q_1,\ldots,q_{r-1},q_r)$. If the message from q_r is missing set $v(q_1,\ldots,q_{r-1},q_r,p)$ to be $\perp_r$.
- At the end of round S, processor p (unless it has previously crashed) has an array of values $v(q_1,\ldots,q_S,p)$. Now p should form $W(q_1,\ldots,q_S,p)$ as the multiset with a single entry $v(q_1,\ldots,q_S,p)$.
- For each r decreasing from $S-1$ to 1

- for each choice of $q_1,\ldots,q_r$, processor p should form a multiset

  $$W\,(q_1,\ldots,q_r,*,\ldots,*,p) = red_{(n-2t)^{S-r-1}t} \cup_{q_{r+1}=1}^{n} W\,(q_1,\ldots,q_r,q_{r+1},*,*,\ldots,*,p)$$

  where in every case the asterisks fill places so that there are $S+1$ entries, either asterisks or indices, to name each multiset.

- Now let p compute $W\,(p) = \cup_{q_1=1}^{n} W\,(q_1,*,\ldots,*,p)$.

- Finally processor p (unless it has crashed) must choose its final value to be $w\,(p) = mid_{(n-2t)^{S-1}t}\,(W\,(p))$.

The algorithm has two phases. First there are S rounds of communication, in each of which each active processor broadcasts all the information it holds and collects the information sent to it. After round r, processor p has an array of values $\langle v(q_1,\ldots,q_r,p)$: each $q_i = 1,\ldots,n\rangle$ where $v(q_1,\ldots,q_r,p)$ is the value p received from q_r representing the initial value $v(q_1)$ as transmitted by q_1 to q_2 in round 1, then relayed by q_2 to q_3 in round 2, and so on. In the second phase, after all communication has occurred, processor p builds for each choice of $q_1,\ldots,q_r$ a multiset $W(q_1,\ldots,q_r,*,\ldots,*,p)$ out of the collection of values $\{v(q_1,\ldots,q_r,q_{r+1},\ldots,q_S,p) : q_j \in \{1,\ldots,n\}$ for $j > r\}$. Now if $q_r,q_{r+1},\ldots,q_S$ are all non-faulty then $v(q_1,\ldots,q_r,q_{r+1},\ldots,q_S,p) = v(q_1,\ldots,q_r)$. In fact the method of constructing $W(q_1,\ldots,q_r,*,\ldots,*,p)$ by successively combining multisets and removing extreme values is designed to ensure that $W(q_1,\ldots,q_r,*,\ldots,*,p)$ is a multiset of size $(n-2t)^{S-r}$ that is a good representative for $v(q_1,\ldots,q_r)$ in that, so long as p does not crash and thus all multisets mentioned are defined,

(i) if q_r has not failed before the start of round $r+1$ then every entry of $W(q_1,\ldots,q_r,*,\ldots,*,p)$ has value $v(q_1,\ldots,q_r)$, and

(ii) the multisets $W(q_1,\ldots,q_r,*,\ldots,*,p_0)$ and $W(q_1,\ldots,q_r,*,\ldots,*,p_1)$ are not very different – in fact they are the same unless q_r failed precisely during round r, in which case they differ in at most $l_{r+1}\cdots l_S$ entries, where l_j denotes the number of processors failing precisely in round j.

These properties are easily proved by descending induction using the recursive construction of $W(q_1,\ldots,q_r,*,\ldots,*,p)$ and using the lemmas about the red_k operators. Finally using these facts about the multisets $W(q_1, *, \ldots, *, p)$ and the property of the operator mid_k we establish that $w(p)$ lies in the range $\rho(U)$ and that

$$|w(p_0) - w(p_1)| \leq \frac{l_1 \cdots l_S}{(n-2t)^S} \cdot \delta(U)$$

which shows that

$$K \leq \frac{\sup\{l_1 l_2 \cdots l_S : l_1 + \cdots l_S \leq t, \text{each } l_i \text{ a non-negative integer}\}}{(n-2t)^S}$$

as the processors that fail precisely in round i are different from those that fail precisely in round j if $i \neq j$.

The above argument hinges on the fact that a faulty processor can cause different correct processors to receive different information only during one round (the round when the faulty processor crashes) since before the crash the faulty processor sends the same correct message to everyone, and after the crash it sends nothing to everyone. The difficulty we face in the failure-by-omission and Byzantine models is that a faulty processor may cause differences between the views held by correct processors in more than one round. To overcome this, in the algorithms of §4 and §8 each processor performs fault detection, examining the messages relayed to it by other processors that they received from q to try to deduce if q is faulty. Once a processor p has deduced that q is faulty, it refuses to listen to messages from q, using $\perp_r$ in place of the values in them. If a processor q_r has not been detected as faulty by everyone by the end of round $r+1$, its performance in round r must have been quite close to correct, and our algorithms remove enough extreme values in forming the multisets $W(q_1, \ldots, q_r, *, \ldots, *, p)$ that these multisets are the same for different p. On the other hand if q_r was detected as faulty by everyone before round r then everyone was ignoring values transmitted by q_r in round r, and the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p)$ will contain only $\perp_r$ and so be the same for different p. Thus the fault detection ensures that a faulty processor can cause significant differences in the views of correct processors only in one round, namely the round before the one in which the last of the other processors detects the failure.

The algorithms of §6 and §8 also obtain better performance than the basic algorithm above by using the operators $chop_k^r$ and $center_k$ which are more complicated than red_k and mid_k but are specially adapted to the situations where the only differences between multisets

$W(q_1, \ldots, q_r, *, \ldots, *, p_0)$ and $W(q_1, \ldots, q_r, *, \ldots, *, p_1)$ are due to replacing a value by $\perp_r$ (unlike the Byzantine case where one value can be replaced by another).

4 The Synchronous Byzantine Failure Model: The Algorithm

Throughout this section of the paper, we require $n > 4t$.

An overview — During each round of communication a correct processor p broadcasts information it holds in the array $\tilde{v}(p_1, \ldots, p_{r-1}, p)$, collects the information sent to it in an array $v(p_1, \ldots, p_r, p)$, tries to deduce which processors are faulty, and then modifies the information it received from processors known to be faulty to form the new array $\tilde{v}(p_1, \ldots, p_r, p)$. The only method a correct processor p uses to detect that process q is faulty is to examine the n values which reach p representing some information that was broadcast by q and then relayed to p by each recipient. If q were correct then every processor would have received the same value in the broadcast and then the correct processors (at least $n - t$ of them) would all have sent the same value to p. Thus if p finds fewer than $n - t$ values the same among the n it received, it can deduce that q was faulty. After the S rounds of communication, a correct processor will have an array of n^S values to operate on. In S steps this array is used to form a collection of $(n - 2t)(n - 4t)^{S-1}$ values by repeatedly removing extreme values from subcollections and then combining subcollections. Finally this collection of values is averaged to give the processor's new value.

In detail, processor p, if correct, must perform the following –

- Set $\tilde{v}(p) = v(p)$.
- In round 1:
  - Broadcast $\tilde{v}(p)$, and denote by $v(q_1, p)$ the value received by p from q_1 purporting to be $\tilde{v}(q_1)$. If the mesage from q_1 is missing or malformed set $v(q_1, p)$ to be $\perp_1$.
  - Set *Fault* $(p, 1)$ to be the empty set.
  - Set $\tilde{v}(q_1, p) = v(q_1, p)$.

- In round r, for $r = 2, \ldots, S$, processor p will start with an array of n^{r-1} values $\langle \tilde{v}(q_1, q_2, \ldots, q_{r-1}, p) : \text{each } q_i = 1, \ldots, n \rangle$ and a set $\textit{Fault}(p, r-1)$ of processors already detected as faulty by p. Now p should
  - Broadcast the array $\langle \tilde{v}(q_1, q_2, \ldots, q_{r-1}, p) \rangle$.
  - Denote by $v(q_1, \ldots, q_{r-1}, q_r, p)$ the value received by p from q_r purporting to be $\tilde{v}(q_1, \ldots, q_{r-1}, q_r)$. If the message from q_r is missing or malformed set $v(q_1, \ldots, q_r, p)$ to be $\perp_r$.
  - For every choice of indices $q_1, \ldots, q_{r-1}$, consider the multiset $\{v(q_1, \ldots, q_{r-1}, 1, p), v(q_1, \ldots, q_{r-1}, 2, p), \ldots, v(q_1, \ldots, q_{r-1}, n, p)\}$. If the most frequently occurring element has multiplicity less than $n - t$, say that "q_{r-1} has been detected as faulty by p in round r." (Note that when $r > 2$, several choices of $q_1, \ldots, q_{r-2}$ may lead to the same q_{r-1} being detected.)
  - Set $\textit{Fault}(p, r) = \textit{Fault}(p, r-1) \cup \{q : q \text{ has been detected as faulty by } p \text{ in round } r\}$.
  - Set $\tilde{v}(q_1, \ldots, q_{r-1}, q_r, p) = \begin{cases} v(q_1, \ldots, q_{r-1}, q_r, p) & \text{if } q_r \notin \textit{Fault}(p, r) \\ \perp_r & \text{if } q_r \in \textit{Fault}(p, r) \end{cases}$.
- At the end of round S, processor p has an array of values $\tilde{v}(q_1, \ldots, q_S, p)$. Now let $W(q_1, \ldots, q_S, p)$ denote the multiset with a single entry $\tilde{v}(q_1, \ldots, q_S, p)$.
- For each r decreasing from $S - 1$ to 1
  - for each choice of $q_1, \ldots, q_r$, processor p should form a multiset

    $$W(q_1, \ldots, q_r, *, \ldots, *, p) = \textit{red}_{(n-4t)^{S-r-1}2t} \cup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$$

    where in every case the asterisks fill places so that there are $S + 1$ entries, either asterisks or indices, to name each multiset.
- Now put $W(p) = \cup_{q_1=1}^{n} W(q_1, *, \ldots, *, p)$.
- Finally processor p should decide on its final value $w(p) = \textit{mid}_{(n-4t)^{S-1}t}(W(p))$. (Note that the amount of reduction in this case is different from that in previous steps.)

In the algorithm above as a convention we set $\mathit{Fault}(p,0) = \emptyset$, $\mathit{Fault}(p,S+1) = \{1,\ldots,n\} \setminus \mathit{Corr}$. We put $\mathit{Exposed}(r) = \cap_{p\in \mathit{Corr}} \mathit{Fault}(p,r)$ and $l_r = |\mathit{Exposed}(r+1)| - |\mathit{Exposed}(r)| = |\mathit{Exposed}(r+1) \setminus \mathit{Exposed}(r)|$. Thus l_r is the number of processors whose behavior in round r led to them being detected as faulty by every correct processor for the first time at the end of round $r+1$. These are the processors that will cause differences between other processors' views in this algorithm, just as in the basic algorithm of §3 differences were caused by processors that crashed during round r.

The behavior of the algorithm is explained by the following lemma, which shows that the multiset $W(q_1,\ldots,q_r,*,\ldots,*,p)$ is a good representative for $\tilde{v}(q_1,\ldots,q_r)$, in that it often consists entirely of copies of that value, and that only processors in $\mathit{Exposed}(r+1)\setminus\mathit{Exposed}(r)$ will cause differences between the multisets computed by different correct processors to represent the same round r value.

Lemma 9 *In an execution of the algorithm of this section, for which $f \le t$, we can conclude:*

(i): *If $p \in \mathit{Corr}$ and $q_r \in \mathit{Corr}$, then all the $(n-4t)^{S-r}$ entries of $W(q_1,\ldots,q_r,*,\ldots,*,p)$ are $\tilde{v}(q_1,\ldots,q_r)$.*

(ii): *If $q_r \notin \mathit{Exposed}(r+1)$, $p_0 \in \mathit{Corr}$, and $p_1 \in \mathit{Corr}$, then $W(q_1,\ldots,q_r,*,\ldots,*,p_0) = W(q_1,\ldots,q_r,*,\ldots,*,p_1)$.*

(iii): *If $q_r \in \mathit{Exposed}(r)$, $p_0 \in \mathit{Corr}$, and $p_1 \in \mathit{Corr}$, then $W(q_1,\ldots,q_r,*,\ldots,*,p_0) = W(q_1,\ldots,q_r,*,\ldots,*,p_1)$.*

(iv): *If $p_0 \in \mathit{Corr}$ and $p_1 \in \mathit{Corr}$, then*

$$|W(q_1,\ldots,q_r,*,\ldots,*,p_0) \cap W(q_1,\ldots,q_r,*,\ldots,*,p_1)| \ge (n-4t)^{S-r} - l_{r+1}\cdot l_{r+2}\cdots l_S.$$

Proof: First we observe that if $p \in \mathit{Corr}$ and $q \in \mathit{Corr}$, then $q \notin \mathit{Fault}(p,r)$ This is proved by induction on r. If $r=1$, and $p \in \mathit{Corr}$, $q \in \mathit{Corr}$ then $q \notin \mathit{Fault}(p,1)$ as $\mathit{Fault}(p,1) = \emptyset$. Now for arbitrary r suppose $p \in \mathit{Corr}$ and $q \in \mathit{Corr}$. If $q_r \in \mathit{Corr}$ then by the induction hypothesis $q \notin \mathit{Fault}(q_r, r-1)$ and so for any choice of $q_1,\ldots,q_{r-2}$ we see $\tilde{v}(q_1,\ldots,q_{r-2},q,q_r) = v(q_1,\ldots,q_{r-2},q,q_r) = \tilde{v}(q_1,\ldots,q_{r-2},q)$ as q is broadcasting correctly. Also q_r broadcasts correctly so $v(q_1,\ldots,q_{r-2},q,q_r,p) = \tilde{v}(q_1,\ldots,q_{r-2},q,q_r)$. Thus the multiset $\{v(q_1,\ldots,q_{r-2},q,1,p), v(q_1,\ldots,q_{r-2},q,2,p), \ldots, v(q_1,\ldots,q_{r-2},q,n,p)\}$

contains at least $(n-t)$ entries each of which is $\tilde{v}(q_1, \ldots, q_{r-2}, q)$. So q is not detected as faulty by p in round r, but by the induction hypothesis $q \notin \mathit{Fault}(p, r-1)$ so we see $q \notin \mathit{Fault}(p, r)$ as required.

Now we use descending induction on r. First, suppose $r = S$.

(i): If $q_S \in \mathit{Corr}$ and $p \in \mathit{Corr}$, then $W(q_1, \ldots, q_S, p) = \{\tilde{v}(q_1, \ldots, q_S, p)\}$, but $q_S \notin \mathit{Fault}(p, S)$ so $\tilde{v}(q_1, \ldots, q_S, p) = v(q_1, \ldots, q_S, p) = \tilde{v}(q_1, \ldots, q_S)$ since q_S correctly broadcast in round S. Thus $W(q_1, \ldots, q_S, p)$ contains $(n-4t)^0 = 1$ entry with value $\tilde{v}(q_1, \ldots, q_S)$.

(ii): If $q_S \notin \mathit{Exposed}(S+1)$, then by definition of the sets $\mathit{Fault}(q, S+1)$, we must have $q_S \in \mathit{Corr}$ and so, by (i) proved above, if $p_0 \in \mathit{Corr}$ and $p_1 \in \mathit{Corr}$, both $W(q_1, \ldots, q_S, p_0)$ and $W(q_1, \ldots, q_S, p_1)$ contain a single entry with value $\tilde{v}(q_1, \ldots, q_S)$ and so are equal.

(iii): If $q_S \in \mathit{Exposed}(S)$ and $p_0 \in \mathit{Corr}$, then $q_S \in \mathit{Fault}(p_0, S)$ so that $\tilde{v}(q_1, \ldots, q_S, p_0) = \perp_S$ and so $W(q_1, \ldots, q_S, p_0)$ is a multiset with a single entry whose value is $\perp_S$. Similarly $W(q_1, \ldots, q_S, p_1)$ has a single entry with value $\perp_S$, so $W(q_1, \ldots, q_S, p_0) = W(q_1, \ldots, q_S, p_1)$.

(iv): The expression $(n-4t)^{S-r} - l_{r+1} \cdots l_S$ evaluates to $1 - 1 = 0$ if $r = S$ (recall that a product of no numbers has value 1 by convention). Thus it is trivially true that $|W(q_1, \ldots, q_S, p_0) \cap W(q_1, \ldots, q_S, p_1)| \geq (n-4t)^{S-r} - l_{r+1} \cdots l_S$ in this case.

We now prove the lemma for some value of r assuming its truth for $r+1$.

(i): If $q_r \in \mathit{Corr}$ and $p \in \mathit{Corr}$ then for $q_{r+1} \in \mathit{Corr}$, by (i) for $r+1$, the multiset $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ consists of $(n-4t)^{S-r-1}$ entries every one having value $\tilde{v}(q_1, \ldots, q_r, q_{r+1})$. However, since $q_r \in \mathit{Corr}$ (and so $q_r \notin \mathit{Fault}(q_{r+1}, r)$), $\tilde{v}(q_1, \ldots, q_r, q_{r+1}) = v(q_1, \ldots, q_r, q_{r+1}) = \tilde{v}(q_1, \ldots, q_r)$. Thus the combined multiset $\cup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ contains at least $(n-4t)^{S-r-1}(n-t)$ entries each of which is $\tilde{v}(q_1, \ldots, q_r)$, namely $(n-4t)^{S-r-1}$ for each of at least $(n-t)$ q_{r+1} that are in Corr. By Lemma 1, applied with $a = b = \tilde{v}(q_1, \ldots, q_r)$, we have that $W(q_1, \ldots, q_r, *, \ldots, *, p)$ consists of exactly $(n-4t)^{S-r}$ entries all of which have value $\tilde{v}(q_1, \ldots, q_r)$.

(ii): If $q_r \notin \mathit{Exposed}\,(r+1)$ then the multiset $\{\tilde{v}\,(q_1,\dots,q_r,q) : q \in \mathit{Corr}\}$ has its most frequent entry (say v) with multiplicity at least $n-2t$. This is proved by contradiction: suppose that there is a choice of $q_1,\dots,q_r$ so that the multiset $\{\tilde{v}\,(q_1,\dots,q_r,q)$ $: q \in \mathit{Corr}\}$ has its most frequent entry with multiplicity less than $n-2t$. Let $p \in \mathit{Corr}$. For $q \in \mathit{Corr}$, $v\,(q_1,\dots,q_r,q,p) = \tilde{v}\,(q_1,\dots,q_r,q)$ so the multiset $\{v\,(q_1,\dots,q_r,1,p),$ $v\,(q_1,\dots,q_r,2,p),\dots,v\,(q_1,\dots,q_r,n,p)\}$ has its most frequent entry with multiplicity less than $n-t$, and so $q_r \in \mathit{Fault}\,(p,r+1)$, but this holds for all correct p which would contradict $q_r \notin \mathit{Exposed}(r+1)$. Now if $q_{r+1} \in \mathit{Corr}$ and $p_0 \in \mathit{Corr}$, by (i) for $r+1$ as above, $W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_0)$ consists of $(n-4t)^{S-r-1}$ copies of $\tilde{v}\,(q_1,\dots,q_r,q_{r+1})$. Thus in this situation $\cup_{q_{r+1}=1}^{n} W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_0)$ contains at least $(n-4t)^{S-r-1}\,(n-2t)$ entries each of which is v (namely $(n-4t)^{S-r-1}$ for each of $(n-2t)$ different choices of q_{r+1}). By Lemma 1, all $(n-4t)^{S-r}$ entries of $W\,(q_1,\dots,q_r,*,\dots,*,p_0)$ are v. If $p_1 \in \mathit{Corr}$ then similarly $W\,(q_1,\dots,q_r,*,\dots,*,p_1)$ consists of $(n-4t)^{S-r}$ copies of v. So these multisets are equal.

(iii): If $q_r \in \mathit{Exposed}\,(r)$ then for any $q_{r+1} \in \mathit{Corr}$, $q_r \in \mathit{Fault}\,(q_{r+1},r)$, so that $\tilde{v}\,(q_1,\dots,q_r,q_{r+1})$ $=\perp_r$. If $p_0 \in \mathit{Corr}$ we can apply (i) for $r+1$ to deduce that $W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_0)$ consists of $(n-4t)^{S-r-1}$ entries all of which are $\perp_r$, and therefore we see that the multiset $\cup_{q_{r+1}=1}^{n} W(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_0)$ contains at least $(n-4t)^{S-r-1}\,(n-t)$ copies of $\perp_r$. Thus by Lemma 1, the multiset $W(q_1,\dots,q_r,*,\dots,*,p_0)$ consists of exactly $(n-4t)^{S-r}$ copies of $\perp_r$. Similarly, if $p_1 \in \mathit{Corr}$, $W(q_1,\dots,q_r,*,\dots,*,p_1)$ consists of $(n-4t)^{S-r}$ copies of $\perp_r$, so these multisets are equal.

(iv): Suppose $p_0 \in \mathit{Corr}$ and $p_1 \in \mathit{Corr}$. Using (i), (ii) and (iii) applied for $r+1$, we see that $W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_0) = W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_1)$ unless $q_{r+1} \in \mathit{Exposed}\,(r+2) \setminus \mathit{Exposed}\,(r+1)$. By (iv) for $r+1$ we have in the case $q_{r+1} \in \mathit{Exposed}\,(r+2) \setminus \mathit{Exposed}\,(r+1)$ that $|W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_0) \cap W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_1)| \geq (n-4t)^{S-r-1} - l_{r+2}\cdots l_S$. We have therefore

$$\begin{aligned} &|\cup_{q_{r+1}=1}^{n} W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_0) \cap \cup_{q_{r+1}=1}^{n} W\,(q_1,\dots,q_r,q_{r+1},*,\dots,*,p_1)| \\ &\qquad \geq \;\; (n-l_{r+1})\,(n-4t)^{S-r-1} + l_{r+1}\left((n-4t)^{S-r-1} - l_{r+2}\cdots l_S\right) \\ &\qquad = \;\; (n-4t)^{S-r-1}\cdot n - l_{r+1}\cdot l_{r+2}\cdots l_S. \end{aligned}$$

Thus by Lemma 2

$$|W(q_1,\ldots,q_r,*,\ldots,*,p_0)\cap W(q_1,\ldots,q_r,*,\ldots,*,p_1)|\geq (n-4t)^{S-r}-l_{r+1}\cdot l_{r+2}\cdots l_S.$$

Q.E.D.

Theorem 10 *The algorithm of this section has performance*

$$K\leq\frac{\sup\{l_1l_2\cdots l_S : l_1+\cdots+l_S\leq t,\text{each } l_i \text{ a non-negative integer}\}}{(n-4t)^{S-1}(n-2t)}$$

Proof: When we apply Lemma 9 with $r=1$ to any execution such that $f\leq t$, we obtain

(i): If $p\in Corr$ and $q_1\in Corr$, then $W(q_1,*,\ldots,*,p)$ consists of $(n-4t)^{S-1}$ entries all of which are $v(q_1)$.

(ii): If $q_1\notin Exposed(2)$, $p_0\in Corr$, and $p_1\in Corr$ then $W(q_1,*,\ldots,*,p_0)$ $=W(q_1,*,\ldots,*,p_1)$.

(iv): If $p_0\in Corr$ and $p_1\in Corr$, then

$$|W(q_1,*,\ldots,*,p_0)\cap W(q_1,*,\ldots,*,p_1)|\geq (n-4t)^{S-1}-l_2\cdot l_3\cdots l_S.$$

Notice that (iii) tells us nothing as $Exposed(1)=\emptyset$. Now if $p\in Corr$ we see that $\cup_{q_1=1}^{n}W(q_1,*,\ldots,*,p)$ contains at least $(n-t)(n-4t)^{S-1}$ entries in the range $\rho\left(\hat{U}\right)$ spanned by initial values of correct processors, namely the $(n-4t)^{S-1}$ copies of $v(q_1)$ for each correct q_1. Then by Lemma 1, $w(p)$ lies in the range $\rho(\hat{U})$. Suppose that $p_0\in Corr$ and $p_1\in Corr$. Then

$$\begin{aligned}|W(p_0)\cap W(p_1)| &\geq (n-4t)^{S-1}(n-l_1)+l_1\left((n-4t)^{S-1}-l_2\cdots l_S\right)\\ &= n(n-4t)^{S-1}-l_1l_2\cdots l_S\end{aligned}$$

as there are l_1 values of q_1 with $q_1\in Exposed(2)$ and $n-l_1$ values of q_1 with $q_1\notin Exposed(2)$. We can apply Lemma 5 to prove

$$|w(p_0)-w(p_1)|\leq\frac{l_1\cdots l_S}{(n-4t)^{S-1}(n-2t)}\cdot\delta\left(\hat{U}\right).$$

We finally note that as $l_1=|Exposed(2)|$, $l_2=|Exposed(3)|-|Exposed(2)|,\ldots,$ $l_S=$ $|Exposed(S+1)|-|Exposed(S)|$, we have each l_i a non-negative integer and also l_1+l_2+

$\ldots + l_S = |Exposed\,(S+1)| = |Fault| \le t$. This proves that our algorithm has, as claimed, performance

$$K \le \frac{\sup\{l_1 l_2 \cdots l_S : l_1 + \cdots + l_S \le t, \text{each } l_i \text{ a non-negative integer}\}}{(n-4t)^{S-1}(n-2t)}.$$

Q.E.D.

It is interesting to note that for $S = 2$ our algorithm therefore gives an implementation of Crusader's Agreement [D] on each value $v(q)$ — each processor p gets either a value (the common value of $W(q, *, p)$) or else the knowledge that q is faulty, and all the processors that get a value get the same value, which is the right one if q is correct. In fact our implementation has a stronger property, that if any p_0 fails to detect that q is faulty, all those p that do detect it know what value p_0 has chosen.

5 The Synchronous Byzantine Failure Model: A Lower Bound

This section gives a formal account of a lower bound, whose asymptotic form was given in [DLPSW], on achievable performance for any S-round, t-resilient approximate agreement algorithm in the synchronous, Byzantine failure model.

Theorem 11 *An algorithm that performs t-resilient approximate agreement in the synchronous Byzantine failure model using at most S rounds of communication, has performance*

$$K \ge \frac{\sup\{l_1 l_2 \cdots l_S : l_1 + \cdots + l_S \le t\}}{(n+t)^S}.$$

Proof: Any algorithm for solving the S-round approximate agreement problem can be given in the following standard form, called a full information protocol, where all information is exchanged for S rounds and then a computation is performed :

- Set $u(p) = v(p)$.
- In round 1, a processor $p \in Corr$
  - broadcasts $u(p)$,
  - denotes by $u(q_1, p)$ the value received by p from q_1 purporting to be $u(q_1)$. (If no such value is received, p should put $u(q_1, p) = \perp_1$.)

- In round r, for $r = 2, 3, \ldots, S$ a processor $p \in Corr$ starts with an array of n^{r-1} values $\langle u(q_1, \ldots, q_{r-1}, p) : \text{each } q_i = 1, \ldots, n\rangle$. It then
  - broadcasts the array $\langle u(q_1, \ldots, q_{r-1}, p)\rangle$,
  - denotes by $u(q_1, \ldots, q_{r-1}, q_r, p)$ the value received by p from q_r purporting to be $u(q_1, \ldots, q_r)$. (If no such value is received, p should put $u(q_1, \ldots, q_r, p) = \perp_r$.)
- Finally a processor $p \in Corr$ applies a function f to its *view*, the array $\langle u(q_1, \ldots, q_S, p)\rangle$ of n^S values, to produce its new value $w(p)$.

Different algorithms are given by different choices of the function f. Notice that the algorithm of §4, which involves computing and modifying values between rounds of communication, is equivalent to one in the standard form because all the computation and modification can be simulated by each processor after all the information is exchanged. So suppose we are given a function f for which the algorithm satisfies the validity condition. Let $l_1, l_2, \ldots, l_S$ be any positive integers so that $l_1 + \cdots + l_S \leq t$. We introduce the collection of multi-indices $I = (i_1, \ldots, i_S)$ where i_k ranges over the integers from 1 to $m_k = \lceil n/l_k \rceil$. We order the multi-indices lexicographically, that is $(i_1, \ldots, i_S) < (j_1, \ldots, j_S)$ if there is some r so that (i) $i_k \leq j_k$ for $k < r$, and (ii) $i_r < j_r$. The multi-indices are totally ordered in this way (which is described as "last index varies fastest" or "row-by-row") and we denote the successor to I by I++. As examples, when $S = 3$, $m_1 = m_2 = 3$, $m_3 = 4$ we have $(1, 2, 3)++ = (1, 2, 4)$, $(1, 2, 4)++ = (1, 3, 1)$ and $(1, 3, 4)++ = (2, 1, 1)$.

To each multi-index I we assign an array M_I of n^S entries defined by

$$M_I(q_1, q_2, \ldots, q_S) = \begin{cases} 1 & \text{if there is some } r \text{ so that (i) } \lceil q_k/l_k \rceil \leq i_k \text{ for } k < r, \text{ and (ii) } \lceil q_r/l_r \rceil < i_r \\ 0 & \text{otherwise} \end{cases}$$

Thus M_I is formed by partitioning the positions in the array into subblocks of size $l_1 \times l_2 \times \cdots \times l_S$. Every entry in a subblock has the same value, which is either 0 or 1. The subblocks filled with 1's all precede those filled with 0's.

If we arrange the arrays M_I in the order of the multi-indices I we get a chain, which we will show has the property that given any two consecutive arrays M_I and M_{I++}, there is some execution of the broadcasting algorithm with $\delta(U) \leq 1$ and $|Fault| \leq t$ leading to one correct processor p_0 receiving M_I as view while another correct processor p_1 receives M_{I++} as view. For this execution $|w(p_0) - w(p_1)| = |f(M_I) - f(M_{I++})|$, so $K \geq |f(M_I) - f(M_{I++})|$. However if we consider an execution where every processor is correct with initial value 0, we find that every processor will get $M_{(1,1,\ldots,1)}$ as view. In an execution where all correct processors have initial value the same, the validity condition requires them to agree on that same value, so $f\left(M_{(1,\ldots,1)}\right) = 0$. Also we consider an execution where the processors 1,2,. . ., $(m_1 - 1)l_1$ are correct with initial value 1, while processors $(m_1 - 1)l_1 + 1, \ldots, n$ follow the algorithm with initial value 0 during the rounds of broadcasting and then stop without computing anything; notice that the arbitrary behavior allowed to a faulty processor includes the possibility of following the algorithm. In this execution the correct processors will receive $M_{(m_1,1,1,\ldots,1)}$ as their view, and the validity condition requires them to agree on 1 as their new value, so $f\left(M_{(m_1,1,\ldots,1)}\right) = 1$. Since the chain of arrays M_I reaches from $I = (1, \ldots, 1)$ to $I = (m_1, 1, \ldots, 1)$ in $(m_1 - 1)m_2 \cdots m_S$ steps, we get a chain of real numbers $f(M_I)$ reaching from 0 to 1 in $(m_1 - 1)m_2 \cdots m_S$ steps. Thus there is some pair of consecutive values where $|f(M_I) - f(M_{I++})| \geq \dfrac{1}{(m_1 - 1)m_2 \cdots m_S} \geq \dfrac{1}{m_1 m_2 \cdots m_S}$, so $K \geq \dfrac{1}{m_1 \cdots m_S}$. Since $m_k = \lceil n/l_k \rceil \leq (n + l_k)/l_k \leq (n+t)/l_k$,

$$K \geq \frac{l_1 l_2 \cdots l_S}{(n+t)^S}$$

As this is true for any choice of $l_1, \ldots, l_S$ with $l_1 + \cdots + l_S \leq t$ we have the lower bound

$$K \geq \frac{\sup\{l_1 l_2 \cdots l_S : l_1 + \cdots + l_S \leq t\}}{(n+t)^S}$$

to which our algorithm is asymptotic as n increases.

The reader can verify that the following construction gives an execution as required with $M_{(i_1,i_2,\ldots,i_S)}$ as the view for p_0, and $M_{(i_1,\ldots,i_S)++}$ as the view for p_1 : The faulty processors are those p such that there is an r with $\lceil p/l_r \rceil = i_r$. Since for each r at most l_r values of p satisfy this condition, the total number of faulty processors is at most $l_1 + \ldots + l_S \leq t$. Choose p_0 and p_1 from among the correct processors. Let $v(p)$ be 1 if $\lceil p/l_1 \rceil \leq i_1$, and 0 if $\lceil p/l_1 \rceil > i_1$.

- Every processor p, correct or faulty, sets $u(p) = v(p)$.

- In round 1,
  - all processors p, except those where $\lceil p/l_1 \rceil = i_1$, broadcast $u(p)$. The remaining p each send the value $u(p)$ to those q where $\lceil q/l_2 \rceil \leq i_2$, but they send the value 0 to those q where $\lceil q/l_2 \rceil > i_2$.
  - All processors p denote by $u(q_1, p)$ the value received by p from q purporting to be $u(q_1)$.
- In round r for $r = 2, \ldots, S-1$
  - all processors p, except those such that $\lceil p/l_r \rceil = i_r$, correctly broadcast the array $\langle u(q_1, \ldots, q_{r-1}, p) \rangle$. The remaining p form another array with

    $$u'(q_1, \ldots, q_{r-1}, p) = \begin{cases} 0 & \text{if } \lceil q_k/l_k \rceil = i_k \text{ for each } k = 1, \ldots, r-1 \\ u(q_1, \ldots, q_{r-1}, p) & \text{else} \end{cases}.$$

    These p send the array $\langle u(q_1, \ldots, q_{r-1}, p) \rangle$ to those q where $\lceil q/l_{r+1} \rceil \leq i_{r+1}$, but they send the array $\langle u'(q_1, \ldots, q_{r-1}, p) \rangle$ to those q where $\lceil q/l_{r+1} \rceil > i_{r+1}$.
  - All processors p denote by $u(q_1, \ldots, q_{r-1}, q_r, p)$ the value received by p from q_r purporting to be $u(q_1, \ldots, q_{r-1}, q_r)$.
- In the final round S
  - all processors p, except those where $\lceil p/l_S \rceil = i_S$ correctly broadcast the array $\langle u(q_1, \ldots, q_{S-1}, p) \rangle$. The remaining p form another array with

    $$u'(q_1, \ldots, q_{S-1}, p) = \begin{cases} 0 & \text{if } \lceil q_k/l_k \rceil = i_k \text{ for each } k = 1, \ldots, S-1 \\ u(q_1, \ldots, q_{S-1}, p) & \text{else} \end{cases}.$$

    These p send the array $\langle u(q_1, \ldots, q_{S-1}, p) \rangle$ to those q where $q \neq p_0$, but they send the array $\langle u'(q_1, \ldots, q_{S-1}, p) \rangle$ to p_0.
  - All processors p denote by $u(q_1, \ldots, q_{S-1}, q_S, p)$ the value received by p from q_S purporting to be $u(q_1, \ldots, q_{S-1}, q_S)$.
- Only the correct processors now calculate their new value from their view. The others halt.

Q.E.D.

6 The Synchronous Crash-Failure Model: The Algorithm

In this section, we require that $n > t$.

An overview — During each round of communication each processor p broadcasts information it holds in the array $v(p_1,\ldots,p_{r-1},p)$ and collects the information sent to it in an array $v(p_1,\ldots,p_r,p)$. After the S rounds of communication, a processor that has not halted will have an array of n^S values to operate on. In S steps this array is used to form a collection of $n\,(2n-2t)^{S-1}$ values by repeatedly doubling, removing excess values from subcollections and then combining subcollections. Finally the *center* operator is applied to this collection of values to give the processor's new value.

In detail, processor p, until it fails, must perform the following –

- In round 1: Broadcast $v\,(p)$, and denote by $v\,(q_1,p)$ the value received by p from q_1 as $v\,(q_1)$. If the message from q_1 is missing set $v\,(q_1,p)$ to be $\perp_1$.

- In round r, for $r = 2,\ldots,S$, processor p will start with an array of n^{r-1} values $\langle v\,(q_1,q_2,\ldots,q_{r-1},p) : \text{each } q_i = 1,\ldots,n\rangle$. Now p should broadcast the array $\langle v(q_1,q_2,\ldots,q_{r-1},p)\rangle$. Denote by $v\,(q_1,\ldots,q_{r-1},q_r,p)$ the value received by p from q_r which was sent as $v\,(q_1,\ldots,q_{r-1},q_r)$. If the message from q_r is missing set $v\,(q_1,\ldots,q_{r-1},q_r,p)$ to be $\perp_r$.

- At the end of round S, processor p has an array of values $v\,(q_1,\ldots,q_S,p)$. Now let $W\,(q_1,\ldots,q_S,p)$ denote the multiset with a single entry $v\,(q_1,\ldots,q_S,p)$.

- For each r decreasing from $S-1$ to 1

  - for each choice of $q_1,\ldots,q_r$, processor p should form a multiset

    $$W\,(q_1,\ldots,q_r,*,\ldots,*,p) = chop^{r+1}_{(2n-2t)^{S-r-1}t} \cup_{q_{r+1}=1}^{n} W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$$

    where in every case the asterisks fill places so that there are $S+1$ entries, either asterisks or indices, to name each multiset.

- Now p computes $W\,(p) = \cup_{q_1=1}^{n} W\,(q_1,*,\ldots,*,p)$.

- Finally processor p (unless it has previously crashed) must decide on the final value $w(p) = center_{(2n-2t)^{S-1}t}(W(p))$.

In the algorithm above, for each $r = 1, \ldots, S$ let $Fail(r)$ denote the set of processors that have failed before sending any of the messages in round r. Also as a convention we define $Fail(S+1)$ to be $Crash$, the set of processors that failed at any time in the execution. We put $l_r = |Fail(r+1)| - |Fail(r)| = |Fail(r+1) \setminus Fail(r)|$. The behavior of the algorithm is explained by the following lemma, which shows that the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p)$ is a good representative for $v(q_1, \ldots, q_r)$, in that it often consists entirely of copies of that value, and that only processors in $Fail(r+1) \setminus Fail(r)$ can cause different processors to choose different multisets as representatives for a round r value.

Lemma 12 *In any execution of the algorithm of this section, such that $f \leq t$, we can conclude:*

(i): *If $p \notin Crash$ then the value of each of the $(2n-2t)^{S-r}$ entries of $W(q_1, \ldots, q_r, *, \ldots, *, p)$ is either $v(q_1, \ldots, q_r)$ or $\perp_r$.*

(ii): *If $q_r \notin Fail(r+1)$ and $p \notin Crash$ then*

$$mult(v(q_1, \ldots, q_r), W(q_1, \ldots, q_r, *, \ldots, *, p)) = (2n-2t)^{S-r}$$

(iii): *If $q_r \in Fail(r)$ and $p \notin Crash$ then*

$$mult(\perp_r, W(q_1, \ldots, q_r, *, \ldots, *, p)) = (2n-2t)^{S-r}$$

(iv): *If $p_0 \notin Crash$ and $p_1 \notin Crash$, then*

$$\begin{aligned}
&|mult(v(q_1, \ldots, q_r), W(q_1, \ldots, q_r, *, \ldots, *, p_0)) \\
&\quad - mult(v(q_1, \ldots, q_r), W(q_1, \ldots, q_r, *, \ldots, *, p_1))| \\
&\leq l_{r+1} \cdot l_{r+2} \cdots l_S.
\end{aligned}$$

Proof: First we observe that if $p \notin Crash$, then $p \notin Fail(r)$ for $r = 1, \ldots, S+1$. Now we use descending induction on r. First, suppose $r = S$.

(i): The multiset $W(q_1, \ldots, q_S, p)$ consists of $(2n-2t)^0 = 1$ entry with value $v(q_1, \ldots, q_S, p)$. Now if q_S did not crash before sending its round S message to p then by the basic property of the crash-failure model, the value it sent was actually $v(q_1, \ldots, q_S)$ so that $v(q_1, \ldots, q_S, p) = v(q_1, \ldots, q_S)$. On the other hand, if q_S crashed before it could send its round S message to p then $v(q_1, \ldots, q_S, p) = \perp_S$.

(ii): If $q_S \notin \mathit{Fail}(S+1)$, then q_S did not crash. As noted above this means that $v(q_1, \ldots, q_S, p) = v(q_1, \ldots, q_S)$ and therefore $W(q_1, \ldots, q_S, p)$ consists of a single entry with value $v(q_1, \ldots, q_S)$.

(iii): If $q_S \in \mathit{Fail}(S)$ then q_S crashed before sending any round S message. As noted above, in this case $v(q_1, \ldots, q_S, p) = \perp_S$ and so $W(q_1, \ldots, q_S, p)$ consists of a single entry with value $\perp_S$.

(iv): Since $l_{r+1} \cdots l_S$ evaluates to 1 when $r = S$ (as an empty product), and each of $W(q_1, \ldots, q_S, p)$ and $W(q_1, \ldots, q_S, p_1)$ have only one entry, the statement

$$\begin{aligned} &|\mathit{mult}(v(q_1, \ldots, q_r), W\,(q_1, \ldots, q_r, *, \ldots, *, p_0)) \\ &\qquad - \mathit{mult}(v(q_1, \ldots, q_r), W\,(q_1, \ldots, q_r, *, \ldots, *, p_1))| \\ &\leq l_{r+1} \cdot l_{r+2} \cdots l_S \end{aligned}$$

is trivially true when $r = S$.

We now prove the lemma for some value of r assuming its truth for $r + 1$.

(i): For each q_{r+1} that has not crashed before the start of round $r + 1$, we know that $v(q_1, \ldots, q_r, q_{r+1})$ is either $v(q_1, \ldots, q_r)$ or $\perp_r$ depending on whether q_r sent its round r message to q_{r+1}. By (i) for $r+1$ we know that $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ consists of $(2n - 2t)^{S-r-1}$ entries each being one of $v(q_1, \ldots, q_r)$, $\perp_r$ or $\perp_{r+1}$. If q_{r+1} crashed before the start of round $r+1$ (so that $v(q_1, \ldots, q_{r+1})$ may be meaningless) then by (i) and (iii) for $r+1$ we know that $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ consists of $(2n-2t)^{S-r-1}$ entries all being $\perp_{r+1}$. Thus $\cup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ consists of $n(2n - 2t)^{S-r-1}$ entries each of which is $v(q_1, \ldots, q_r)$, $\perp_r$ or $\perp_{r+1}$. Also there are at least $(n - t)(2n - 2t)^{S-r-1}$ entries that are not $\perp_{r+1}$, namely all those coming from the at least $n - t$ values of q_{r+1} that are not in $\mathit{Fail}(r + 2)$ (by (ii) for $r + 1$). Thus when we apply chop_k^{r+1}, where $k = t(2n-2t)^{S-r-1}$, to $\cup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$, we will remove every occurrence of $\perp_{r+1}$ and be left with $(2n-2t)^{S-r}$ entries all either $v(q_1,\ldots,q_r)$ or $\perp_r$.

(ii): If $q_r \notin \mathit{Fail}(r+1)$ then q_r sends all its round r messages and so every q_{r+1} that has not failed before starting round $r+1$ has $v(q_1,\ldots,q_r,q_{r+1}) = v(q_1,\ldots,q_r)$ and so by (i) for $r+1$, $W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ will consist only of copies of $v(q_1,\ldots,q_r)$ and of $\perp_{r+1}$. If q_{r+1} has failed before starting round $r+1$ then by (iii) for $r+1$ $W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ contains only copies of $\perp_{r+1}$. Thus the combined multiset $\cup_{q_{r+1}=1}^{n} W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ contains only copies of $v(q_1,\ldots,q_r)$ and of $\perp_{r+1}$, and so the same will be true of $W(q_1,\ldots,q_r,*,\ldots,*,p)$ which is the result of applying chop_k^{r+1}, where $k = t(2n-2t)^{S-r-1}$. Combined with what we proved in (i), this gives that $W(q_1,\ldots,q_r,*,\ldots,*,p)$ consists of exactly $(2n-2t)^{S-r}$ copies of $v(q_1,\ldots,q_r)$.

(iii): If $q_r \in \mathit{Fail}(r)$ then q_r sent no messages in round r, so that every q_{r+1} that has not crashed before starting round $r+1$ has $v(q_1,\ldots,q_r,q_{r+1}) = \perp_r$ and so by (i) for $r+1$, $W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ will consist only of copies of $\perp_r$ and of $\perp_{r+1}$. If q_{r+1} has crashed before starting round $r+1$ then by (iii) for $r+1$ $W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ contains only copies of $\perp_{r+1}$. Thus the multiset $\cup_{q_{r+1}=1}^{n} W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ contains only copies of $\perp_r$ and of $\perp_{r+1}$, and so the same will be true of the multiset $W(q_1,\ldots,q_r,*,\ldots,*,p)$ which is the result of applying chop_k^{r+1}, where $k = t(2n-2t)^{S-r-1}$. Combined with what we proved in (i), this gives that $W(q_1,\ldots,q_r,*,\ldots,*,p)$ consists of exactly $(2n-2t)^{S-r}$ copies of $\perp_r$.

(iv): We have that $W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_0) = W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_1)$ unless $q_{r+1} \in \mathit{Fail}(r+2)\setminus\mathit{Fail}(r+1)$, by (ii) and (iii) for $r+1$. For the other l_{r+1} values of q_{r+1} we have by (iv) for $r+1$ that

$$\begin{aligned}
&|\mathit{mult}(v(q_1,\ldots,q_r,q_{r+1}), W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_0)) \\
&\quad -\mathit{mult}(v(q_1,\ldots,q_r,q_{r+1}), W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_1))| \\
&\leq l_{r+2}\cdot l_{r+3}\cdots l_S
\end{aligned}$$

For each of these q_{r+1}, $v(q_1,\ldots,q_r,q_{r+1})$ is either $v(q_1,\ldots,q_r)$ or $\perp_r$, so we see

$$\begin{aligned}
&|mult(v(q_1,\ldots,q_r),\cup_{q_{r+1}=1}^{n}W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_0))\\
&\quad -mult(v(q_1,\ldots,q_r),\cup_{q_{r+1}=1}^{n}W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_1))|\\
&\;+|mult(\perp_r,\cup_{q_{r+1}=1}^{n}W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_0))\\
&\quad -mult(\perp_r,\cup_{q_{r+1}=1}^{n}W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_1))|\\
&\le l_{r+1}\cdot l_{r+2}\cdots l_S
\end{aligned}$$

With this bound and the facts in (i) we can apply Lemma 3 to complete the proof.

Q.E.D.

Theorem 13 *The algorithm of this section has performance*

$$K \le \frac{\sup\{l_1 l_2\cdots l_S : l_1+\cdots+l_S\le t, \text{each } l_i \text{ a non-negative integer}\}}{(2n-2t)^S}.$$

Proof: We have by (ii) and (iii) of Lemma 12 for $r=1$ that $W(q_1,*,\ldots,*,p_0) = W(q_1,*,\ldots,*,p_1)$ unless $q_1\in Fail(2)\backslash Fail(1)$. For these l_1 values of q_1 we have by (iv) for $r=1$ that

$$|mult(v(q_1),W\,(q_1,*,\ldots,*,p_0)-mult(v(q_1),W\,(q_1,*,\ldots,*,p_1)\,|\le l_2\cdot l_3\cdots l_S.$$

We can apply Lemma 6 with $V=\cup_{q_1=1}^{n}W(q_1,*,\ldots,*,p_0)$, $W=\cup_{q_1=1}^{n}W(q_1,*,\ldots,*,p_1)$, $N=n(2n-2t)^{S-1}$, $m=l_1\cdots l_S$, $k=t(2n-2t)^{S-1}$ and $[a,b]=\rho(U)$ to prove that each of $w(p_0)=center_k(V)$ and $w(p_1)=center_k(W)$ lie in $\rho(U)$ and that

$$|w\,(p_0)-w\,(p_1)|\le\frac{l_1\cdots l_S}{(2n-2t)^S}\cdot\delta\,(U)\,.$$

We finally note that as $l_1=|Fail\,(2)|-|Fail(1)|$, $l_2=|Fail\,(3)|-|Fail\,(2)|$,..., $l_S=|Fail\,(S+1)|-|Fail\,(S)|$, we have each l_i a non-negative integer and also $l_1+l_2+\ldots+l_S=|Fail\,(S+1)|-|Fail(1)|\le t$. This proves that our algorithm has, as claimed, performance

$$K\le\frac{\sup\{l_1l_2\cdots l_S : l_1+\cdots l_S\le t, \text{each } l_i \text{ a non-negative integer}\}}{(2n-2t)^S}.$$

Q.E.D.

It is interesting to note that in any execution where there is a round r such that some $l_r=0$ (this means that in round r no new processors crashed) then $K=0$ (so exact agreement is obtained). It is proved in [DM] that processors can have common knowledge of exact agreement only if there is such a round.

7 The Synchronous Crash-Failure Model: A Lower Bound

This section gives a formal account of a new lower bound on achievable performance for any S-round approximate agreement algorithm in the crash-failure model. Any algorithm for solving the S-round approximate agreement problem can be given in the form of a full information protocol (as in §5) where all information is exchanged for S rounds giving each processor p a view $\langle v(q_1, \ldots, q_S, p)\rangle$ and then p applies a function f to the view to give its new value $w(p)$. For the remainder of this section we consider a fixed full information protocol.

To prove a lower bound on the performance achievable we are going to construct a chain of views as in §5, but this time we will do so implicitly by giving a recursive recipe for the execution that lies between successive views. This proof is very closely related to the proofs in [DM] and [MT] of the impossibility of exact agreement in fewer than $t+1$ rounds, and also to the proof in [CD] of the impossibility of simultaneous firing in fewer than $t+1$ rounds. An execution in the crash-failure model is very easy to describe — we need only specify the initial value of each processor and say which processors failed in each round and which messages they sent in that round. We say that two executions ρ and ρ' are ***directly similar*** (written $\rho \approx \rho'$) if some processor p is correct in each and obtains the same view in each. We say similarly that ρ and ρ' are ***k-similar*** (written $\rho \sim^k \rho'$) if there are $k+1$ executions $\rho_0, \rho_1, \ldots, \rho_k$ so that $\rho_0 = \rho$, $\rho_k = \rho'$, and $\rho_i \approx \rho_{i+1}$ for each i. Thus $\sim^1$ is just $\approx$, and if $\rho \sim^k \rho'$ and $\rho' \sim^m \rho''$ then $\rho \sim^{k+m} \rho''$. Note that $\rho \sim^k \rho'$ implies $\rho' \sim^k \rho$ and $\rho \sim^m \rho'$ for $m \geq k$.

Let $l_1, l_2, \ldots, l_S$ be any collection of positive integers such that $l_1 + \ldots + l_S \leq t$. Put $m_i = \lceil n/l_i \rceil$. We have

Lemma 14 *Let $1 \leq r \leq S-1$. Let $\rho = \rho_0$ be an execution of the protocol such that no failures occur after the end of round r, and the number of failures by the end of round i is at most $l_1 + \ldots + l_i$ for any i. Denote by $\hat{\rho}$ the execution that is identical to ρ for the first $r-1$ rounds but has no failures during any later round. Then $\rho \sim^{N(r)} \hat{\rho}$ where $N(r) = \sum_{i=r+1}^{S} \prod_{j=r+1}^{i} 2m_j + 1$.*

Proof: Let the processors that fail in round r in ρ be denoted $i_1, \ldots, i_m$. We will use descending induction on r. So suppose $r = S - 1$. (Note that the statement is not true if $r = S$.) For each $k = 1,\ldots,m_S$ let q_k be the greatest processor index that is not among the processors that failed in ρ nor in the range $(k-2)l_S + 1,\ldots,kl_S$, and let p_k be the least processor index that is not among the processors that failed in ρ, nor in the range $(k-2)l_S + 1,\ldots,(k-1)l_S$. Then clearly $p_k < q_{k-1}$ and also $p_k < q_k$. Let ρ_{2k-1} denote the execution that is identical to ρ during the first $S-2$ rounds, and then also during round $S-1$ except that the processors $i_1,\ldots,i_m$ do send to any processor with index $1,2,\ldots,(k-1)l_S$ as well as those processors that they send to in ρ. In round S, each of the processors $(k-1)l_S + 1,\ldots,kl_S$ that has not failed earlier, fails after sending messages to processors $1,\ldots,q_k - 1$. The assumptions on failure numbers in ρ mean that this execution involves at most t failures. Also let ρ_{2k} denote the execution identical to ρ during the first $S-2$ rounds, and then also during round $S-1$ except that the processors $i_1,\ldots,i_m$ do send to any processor with index $1,2,\ldots,kl_S$ as well as those processors that they send to in ρ. In round S, each of the processors $(k-1)l_S + 1,\ldots,kl_S$ that has not failed earlier, fails after sending messages to processors $1,\ldots,q_k - 1$. The assumptions on failure numbers in ρ mean that this execution involves at most t failures. Clearly the view of p_k is the same in $\rho_{2(k-1)}$ as in ρ_{2k-1} so $\rho_{2(k-1)} \approx \rho_{2k-1}$. Similarly the view of q_k is the the same in ρ_{2k-1} as in ρ_{2k} so $\rho_{2k-1} \approx \rho_{2k}$. Also let $\tilde{\rho}$ denote the execution identical to ρ during the first $S-2$ rounds with no failures during round $S-1$ and in round S each of the processors $i_1,i_2,\ldots,i_m$ as well as each of $(m_S - 1)l_S + 1,\ldots,m_S l_S$ that hasn't failed earlier, fails after sending messages to processors $1,\ldots,q_{m_S} - 1$. The view of q_{m_S} is the same in $\tilde{\rho}$ as in $\rho_{2m_S - 1}$ so $\rho_{2m_S-1} \approx \tilde{\rho}$. Similarly the view of p_{m_S} is the same in $\tilde{\rho}$ as in $\hat{\rho}$ so $\tilde{\rho} \approx \hat{\rho}$. Thus examining the whole argument, $\rho \sim^{2m_S+1} \hat{\rho}$.

Now we assume we have the result for $r+1$ and prove it for r. For each $k = 1,\ldots,m_{r+1}$ we let ρ_{3k-2} denote the execution identical to ρ for the first $r-1$ rounds and also in round r except that the processors $i_1,\ldots,i_m$ do send to any processor with index $1,2,\ldots,(k-1)l_{r+1}$ as well as those processors that they send to in ρ. In round $r+1$, each of the processors $(k-1)l_{r+1} + 1,\ldots,kl_{r+1}$ that has not failed earlier, fails before sending any messages. No failures occur after round $r+1$. The assumptions on the number of failures in ρ imply that this execution also satisfies those assumptions. We let ρ_{3k-1} denote the execution identical to ρ for the first $r-1$ rounds and also in round r except that the processors $i_1,\ldots,i_m$ do send to any processor with index $1,2,\ldots,kl_{r+1}$ as well as those processors that they send to in ρ. In round $r+1$, each of the processors $(k-1)l_{r+1}+1,\ldots,kl_{r+1}$ that has not failed earlier, fails before sending any messages. No failures occur after round $r+1$. The assumptions on the number of failures in ρ imply that this execution also satisfies those assumptions. We let ρ_{3k} denote the execution identical to ρ for the first $r-1$ rounds and also in round r except that the processors $i_1,\ldots,i_m$ do send to any processor with index $1,2,\ldots,kl_{r+1}$ as well as those processors that they send to in ρ. No failures occur after round r. The assumptions on the number of failures in ρ imply that this execution also satisfies those assumptions. Now by the lemma for $r+1$ we have $\rho_{3(k-1)} \sim^{N(r+1)} \rho_{3k-2}$ and $\rho_{3k-1} \sim^{N(r+1)} \rho_{3k}$. Also every processor gets the same view in ρ_{3k-2} as in ρ_{3k-1} so $\rho_{3k-2} \approx \rho_{3k-1}$. Further $\rho_{3m_{r+1}}$ in which processors $i_1,\ldots,i_m$ fail at the very end of round r can also be viewed as an execution in which they fail at the very start of round $r+1$, and so by the lemma for $r+1$ we have $\rho_{3m_{r+1}} \sim^{N(r+1)} \hat{\rho}$. Putting all these pieces of chain together we see $\rho \sim^{(2m_{r+1}+1)N(r+1)+m_{r+1}}$ $\hat{\rho}$, but $(2m_{r+1}+1)N(r+1)+m_{r+1} \leq (2m_{r+1}+1)(N(r+1)+1) = N(r)$. Q.E.D.

Theorem 15 *Any algorithm that solves t-resilient approximate agreement for the synchronous crash-failure model in at most S rounds has performance*

$$K \geq \frac{\sup\,(l_1\cdots l_S : l_1+\cdots+l_S \leq t,\ \text{each } l_i \text{ a nonnegative integer})}{(2n+3t)^S}$$

Proof: We prove that if $\rho = \rho_0$ is the execution where all processors have initial value 0 and no failures occur, and $\hat{\rho}$ is the execution where all initial values are 1 and no failures occur, then $\rho \sim^N \hat{\rho}$ where $N \leq (2m_1+2)\cdot(2m_2+2)\cdots(2m_S+2)$. We will give separate proofs if $S>1$ and $S=1$. First suppose $S>1$. For each $k=1,\ldots,m_1$ let ρ_{3k-2} denote the execution where processors $1,\ldots,(k-1)l_1$ have initial value 1, and the others have initial value 0, and where processors $(k-1)l_1+1,\ldots,kl_1$ fail in round 1 before sending any messages, but no other failures occur. Let ρ_{3k-1} denote the execution where processors $1,\ldots,kl_1$ have initial value 1, and the others have initial value 0, and where processors $(k-1)l_1+1,\ldots,kl_1$ fail in round 1 before sending any messages, but no other failures occur. Let ρ_{3k} denote the execution where processors $1,\ldots,kl_1$ have initial value 1, and the others have initial value 0, and where no failures occur. By Lemma 14, $\rho_{3(k-1)} \sim^{N(1)} \rho_{3k-2}$ and $\rho_{3k-1} \sim^{N(1)} \rho_{3k}$.

Also the view of every processor is the same in ρ_{3k-2} as in ρ_{3k-1} since the initial value of a processor that fails before sending any message is irrelevant, and so $\rho_{3k-2} \approx \rho_{3k-1}$. Since $\rho_{3m_1} = \hat{\rho}$, we have $\rho \sim^N \hat{\rho}$ where $N = 2m_1(N(1)+1) \leq \prod_{j=1}^{S} 2m_j + 2$ as we see by writing $2m_j + 2$ as $(2m_j+1)+1$ and expanding the product. In the case $S = 1$ for each $k = 1,\ldots,m_1$ let q_k be the greatest processor index that is not in the range $(k-2)l_1 + 1,\ldots,kl_1$. Let p_k be the least processor index that is not in the range $(k-2)l_1 + 1,\ldots,(k-1)l_1$. Thus $p_k < q_k$ and $p_k < q_{k-1}$. Let ρ_{2k-1} denote the execution in which the processors with index $1, 2,\ldots,(k-1)l_1$ have initial value 1 and the others have initial value 0 and in round 1, each of the processors $(k-1)l_1 + 1,\ldots,kl_1$ fails after sending messages to processors $1,\ldots,q_k - 1$. Let ρ_{2k} denote the execution in which the processors with index $1, 2,\ldots,kl_1$ have initial value 1 and the others have initial value 0 and in round 1, each of the processors $(k-1)l_1 + 1,\ldots,kl_1$ fails after sending messages to processors $1,\ldots,q_k - 1$. The view of p_k is the same in $\rho_{2(k-1)}$ as in ρ_{2k-1} so $\rho_{2(k-1)} \approx \rho_{2k-1}$. Similarly the view of q_k is the same in ρ_{2k-1} as in ρ_{2k} so $\rho_{2k-1} \approx \rho_{2k}$. As the view of p_{m_1} is the same in ρ_{2m_1} as in $\hat{\rho}$ we have that $\rho_{2m_1} \approx \hat{\rho}$, and so $\rho \sim^N \hat{\rho}$, where $N = 2m_1 + 1 \leq 2m_1 + 2$.

Now we have shown how to construct a sequence $\rho_0 = \rho, \rho_1,\ldots,\rho_N = \hat{\rho}$ where $\rho_i \approx \rho_{i+1}$, that is there is some processor p_i whose view (which we will call M_i) is the same in ρ_i and in ρ_{i+1}. Since M_0 is a view in a failure-free execution where every initial value is 0 we must have $f(M_0) = 0$. Similarly M_{N-1} is a view in a failure-free execution where all initial values are 1 so $f(M_{N-1}) = 1$. Thus there must be some i so that $|f(M_i) - f(M_{i+1})| \geq 1/N$ but each of M_i and M_{i+1} are views in the execution ρ_{i+1} which from the construction clearly has all initial values either 0 or 1. Thus we have proved that any algorithm has $K \geq 1/N$. Since $N \leq \prod_{j=1}^{S} 2m_j + 2 \leq \prod_{j=1}^{S}(2n/l_j + 3)$ we have $K \geq \prod_{j=1}^{S} l_j/(2n + 3l_j) \geq \prod_{j=1}^{S} l_j/(2n+3t)$. As $l_1,\ldots,l_S$ were arbitrary, subject only to $l_1 + \ldots + l_S \leq t$, we have

$$K \geq \frac{\sup\,(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ each } l_i \text{ a nonnegative integer})}{(2n+3t)^S}.$$

Q.E.D.

8 The Synchronous Failure-By-Omission Model

In this section, we require $n > 2t$.

An overview — During each round of communication each processor p broadcasts information it holds in the array $\tilde{v}(p_1, \ldots, p_{r-1}, p)$, collects the information sent to it in an array $v(p_1, \ldots, p_r, p)$, tries to deduce which processors are faulty, and then modifies the information it received from processors known to be faulty to form the new array $\tilde{v}(p_1, \ldots, p_r, p)$. The only method a processor p uses to detect that process q is faulty, is to examine the n values which reach p representing some information that was broadcast by q and then relayed to p by each recipient. If q were correct then no active processor would have failed to receive q's value in the broadcast and so none of the values reaching p would be $\perp_{r-1}$. Thus if p finds any entry being $\perp_{r-1}$ among those it received, it can deduce that q was faulty. After the S rounds of communication, a processor will have an array of n^S values to operate on. In S steps this array is used to form a collection of $(2n-4t)^{S-1}(2n-2t)$ values by repeatedly removing extreme values from subcollections and then combining subcollections. Finally this collection of values is averaged to give the processor's new value.

In detail, the algorithm requires processor p to perform the following (unless it has previously crashed) –

- Set $\tilde{v}(p) = v(p)$.
- In round 1:
  - Broadcast $\tilde{v}(p)$, and denote by $v(q_1, p)$ the value received by p from q_1 as $\tilde{v}(q_1)$. If the message from q_1 is missing set $v(q_1, p)$ to be $\perp_1$.
  - Set *Fault* $(p, 1)$ to be the empty set.
  - Set $\tilde{v}(q_1, p) = v(q_1, p)$.
- In round r, for $r = 2, \ldots, S$, processor p will start with an array of n^{r-1} values $\langle \tilde{v}(q_1, q_2, \ldots, q_{r-1}, p) : \text{each } q_i = 1, \ldots, n \rangle$ and a set *Fault* $(p, r-1)$ of processors already detected as faulty by p. Now p should
  - Broadcast the array $\langle \tilde{v}(q_1, q_2, \ldots, q_{r-1}, p) \rangle$.
  - Denote by $v(q_1, \ldots, q_{r-1}, q_r, p)$ the value received by p from q_r as $\tilde{v}(q_1, \ldots, q_{r-1}, q_r)$. If the message from q_r is missing set $v(q_1, \ldots, q_{r-1}, q_r, p)$ to be $\perp_r$.

  - For every choice of indices $q_1, \ldots, q_{r-1}$, consider the multiset $\{v(q_1, \ldots, q_{r-1}, 1, p)$, $v(q_1, \ldots, q_{r-1}, 2, p), \ldots, v(q_1, \ldots, q_{r-1}, n, p)\}$. If any entry is $\perp_{r-1}$ say that "q_{r-1} has been detected as faulty by p in round r". (Note that several choices of $q_1, \ldots, q_{r-2}$ may lead to the same q_{r-1} being detected.)
  - Set $\mathit{Fault}(p, r) = \mathit{Fault}(p, r-1) \cup \{q : q$ has been detected as faulty by p in round $r\}$.
  - Set $\tilde{v}(q_1, \ldots, q_{r-1}, q_r, p) = \begin{cases} v(q_1, \ldots, q_{r-1}, q_r, p) & \text{if } q_r \notin \mathit{Fault}(p, r) \\ \perp_r & \text{if } q_r \in \mathit{Fault}(p, r) \end{cases}$.

- At the end of round S, processor p (unless it has previously crashed) has an array of values $\tilde{v}(q_1, \ldots, q_S, p)$. Now p should let $W(q_1, \ldots, q_S, p)$ denote the multiset with a single entry $\tilde{v}(q_1, \ldots, q_S, p)$.

- For each r decreasing from $S-1$ to 1

  - for each choice of $q_1, \ldots, q_r$, processor p must form a multiset

    $W(q_1, \ldots, q_r, *, \ldots, *, p) = \mathit{chop}^{r+1}_{(2n-4t)^{S-r-1}2t} \cup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$

    where in every case the asterisks fill places so that there are $S+1$ entries, either asterisks or indices, to name each multiset.

- Now put $W(p) = \cup_{q_1=1}^{n} W(q_1, *, \ldots, *, p)$.

- Finally processor p should choose a final value $w(p) = \mathit{center}_{(2n-4t)^{S-1}t}(W(p))$. (Note that the amount of reduction is different from that in previous steps).

In the algorithm above, for each $r = 1, \ldots, S$ let $\mathit{Fail}(r)$ denote the set of processors that have crashed before sending any of the messages in round r. Let $\mathit{Exposed}(r) = \mathit{Fail}(r) \cup \cap_{p \notin \mathit{Fail}(r+1)} \mathit{Fault}(p, r)$. Also as a convention we set $\mathit{Exposed}(S+1) = \{1, \ldots, n\} \setminus \mathit{Corr}$. Note that $\mathit{Exposed}(r) \subset \mathit{Exposed}(r+1)$. We put $l_r = |\mathit{Exposed}(r+1)| - |\mathit{Exposed}(r)| = |\mathit{Exposed}(r+1) \setminus \mathit{Exposed}(r)|$. The behavior of the algorithm is explained by the following lemma, which shows that when p does not crash during execution of the algorithm, the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p)$ is a good representative for $\tilde{v}(q_1, \ldots, q_r)$, in that it often consists only of copies of that value, and that only processors in $\mathit{Exposed}(r+1) \setminus \mathit{Exposed}(r)$ can cause different processors to choose different representatives for a round r value.

Lemma 16 *In any execution of the algorithm of this section, such that $f \leq t$, we can conclude:*

(i): *If $p \notin Crash$ then the value of each of the $(2n-4t)^{S-r}$ entries of $W\,(q_1, \ldots, q_r, *, \ldots, *, p)$ is either $\tilde{v}(q_1, \ldots, q_r)$ or $\perp_r$.*

(ii): *If $q_r \notin Exposed\,(r+1)$ and $p \notin Crash$, then*

$$mult(\tilde{v}(q_1, \ldots, q_r), W\,(q_1, \ldots, q_r, *, \ldots, *, p)) = (2n-4t)^{S-r}.$$

(iii): *If $q_r \in Exposed\,(r)$ and $p \notin Crash$, then*

$$mult(\perp_r, W\,(q_1, \ldots, q_r, *, \ldots, *, p)) = (2n-4t)^{S-r}.$$

(iv): *If $p_0 \notin Crash$ and $p_1 \notin Crash$, then*

$$\begin{aligned} &|mult(\tilde{v}(q_1, \ldots, q_r), W\,(q_1, \ldots, q_r, *, \ldots, *, p_0)) \\ &\qquad - mult(\tilde{v}(q_1, \ldots, q_r), W\,(q_1, \ldots, q_r, *, \ldots, *, p_1))| \\ &\leq l_{r+1} \cdot l_{r+2} \cdots l_S. \end{aligned}$$

Proof: First we observe from the algorithm that $\tilde{v}(q_1, \ldots, q_r, p)$ can never have the value $\perp_j$ for $j > r$. Next we observe that if $p \notin Fail(r+1)$ and $q \in Corr$ then $q \notin Fault(p,r)$. This is proved by induction on r. The case $r = 1$ is trivial as $Fault(p,1)$ is empty. Now for arbitrary r, suppose $p \notin Fail(r+1)$ and $q \in Corr$. Fix $q_1, \ldots, q_{r-2}$. If q_r does not send properly to p in round r (in particular if $q_r \in Fail(r)$) then $v(q_1, \ldots, q_{r-2}, q, q_r, p) = \perp_r$. On the other hand if q_r does send to p in round r then $v(q_1, \ldots, q_{r-2}, q, q_r, p) = \tilde{v}(q_1, \ldots, q_{r-2}, q, q_r) = v(q_1, \ldots, q_{r-2}, q, q_r)$ since by the induction hypothesis $q \notin Fault(q_r, r-1)$, and because q must send correctly $v(q_1, \ldots, q_{r-2}, q, q_r) = \tilde{v}(q_1, \ldots, q_{r-2}, q)$ which as we noted above is not equal to $\perp_{r-1}$. Thus no entry of $\{v(q_1, \ldots, q_{r-2}, q, q_r, p) : q_r = 1, \ldots, n\}$ is $\perp_{r-1}$ proving that $q \notin Fault(p,r)$. Now we use descending induction on r to prove the lemma. First, suppose $r = S$.

(i): The multiset $W\,(q_1, \ldots, q_S, p)$ consists of $(2n-4t)^0 = 1$ entry with value $\tilde{v}(q_1, \ldots, q_S, p)$. Now if q_S did not send its round S message to p or if $q_S \in Fault(p,S)$ then $\tilde{v}(q_1, \ldots, q_S, p) = \perp_S$, while otherwise $\tilde{v}(q_1, \ldots, q_S, p) = v(q_1, \ldots, q_S, p) = \tilde{v}(q_1, \ldots, q_S)$ since in the failure-by-omission model any value that is sent is correct.

(ii): If $q_S \notin \mathit{Exposed}(S+1)$ then $q_S \in \mathit{Corr}$ and so q_S did send its round S message to p and also $q_S \notin \mathit{Fault}(p,S)$. As noted in the discussion in part (i) above this means that $\tilde{v}(q_1,\ldots,q_S,p) = \tilde{v}(q_1,\ldots,q_S)$ and so $W(q_1,\ldots,q_S,p)$ consists of a single entry with value $\tilde{v}(q_1,\ldots,q_S)$.

(iii): If $q_S \in \mathit{Exposed}(S)$ then either $q_S \in \mathit{Fail}(S)$ so q_S did not send its round S message to p, or $q_S \in \mathit{Fault}(p,S)$. In either case as noted in part (i) above, $\tilde{v}(q_1,\ldots,q_S,p) = \perp_S$ and so $W(q_1,\ldots,q_S,p)$ consists of a single entry with value $\perp_S$.

(iv): Since $l_{r+1}\cdots l_S$ evaluates to 1 when $r = S$ (as an empty product), and each of $W(q_1,\ldots,q_S,p_0)$ and $W(q_1,\ldots,q_S,p_1)$ have only one entry, the statement

$$\begin{aligned}&|\mathit{mult}(v(q_1,\ldots,q_r),W(q_1,\ldots,q_r,*,\ldots,*,p_0))\\&\qquad -\mathit{mult}(v(q_1,\ldots,q_r),W(q_1,\ldots,q_r,*,\ldots,*,p_1))|\\&\le l_{r+1}\cdot l_{r+2}\cdots l_S\end{aligned}$$

is trivially true when $r = S$.

We now prove the lemma for some value of r assuming its truth for $r+1$.

(i): For each q_{r+1} that has not crashed before the start of round $r+1$, we know that $v(q_1,\ldots,q_r,q_{r+1})$ is either $\perp_r$ (if q_r failed to send its round r message to q_{r+1} or if $q_r \in \mathit{Fault}(q_r,r)$) or $\tilde{v}(q_1,\ldots,q_r)$ (otherwise). By (i) for $r+1$ we know that $W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ consists of $(2n-4t)^{S-r-1}$ entries each of which is one of $\tilde{v}(q_1,\ldots,q_r)$, $\perp_r$ or $\perp_{r+1}$. If q_{r+1} crashed before the start of round $r+1$ (so that $v(q_1,\ldots,q_{r+1})$ may be meaningless) then $q_{r+1} \in \mathit{Exposed}(r+1)$, so by (iii) for $r+1$ we know that $W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ consists of $(2n-4t)^{S-r-1}$ entries all being $\perp_{r+1}$. Thus $\cup_{q_{r+1}=1}^{n} W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ consists of $n(2n-4t)^{S-r-1}$ entries each of which is $\tilde{v}(q_1,\ldots,q_r)$, $\perp_r$ or $\perp_{r+1}$. Also there are at least $(n-t)(2n-4t)^{S-r-1}$ entries that are not $\perp_{r+1}$, namely all those coming from the at least $n-t$ values of q_{r+1} that are not in $\mathit{Exposed}(r+2)$ (by (ii) for $r+1$). Thus when we apply chop_k^{r+1}, where $k = 2t(2n-4t)^{S-r-1}$, to $\cup_{q_{r+1}=1}^{n} W(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p)$ we will remove every occurrence of $\perp_{r+1}$ and be left with $(2n-4t)^{S-r}$ entries all either $\tilde{v}(q_1,\ldots,q_r)$ or $\perp_r$.

(ii): If $q_r \notin \mathit{Exposed}(r+1)$ then for every $q \in \mathit{Corr}$, $\tilde{v}(q_1, \ldots, q_r, q) = \tilde{v}(q_1, \ldots, q_r)$. This is proved by contradiction: suppose there is $q_{r+1} \in \mathit{Corr}$ with $\tilde{v}(q_1, \ldots, q_r, q_{r+1}) = \perp_r$ and therefore for any $p \notin \mathit{Fail}(r+2)$ we will have $v(q_1, \ldots, q_r, q_{r+1}, p) = \perp_r$ as q_{r+1} broadcasts correctly, and hence p will detect q_r as faulty in round $r+1$. This holding for all $p \notin \mathit{Fail}(r+2)$ contradicts the assumption $q_r \notin \mathit{Exposed}(r+1)$. Now by (ii) for $r+1$, if $q_r \notin \mathit{Exposed}(r+1)$, $q_{r+1} \in \mathit{Corr}$ and $p \notin \mathit{Crash}$, then $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ consists of $(2n-4t)^{S-r-1}$ entries all with value $\tilde{v}(q_1, \ldots, q_r, q_{r+1}) = \tilde{v}(q_1, \ldots, q_r)$. Hence if $q_r \notin \mathit{Exposed}(r+1)$ and $p \notin \mathit{Crash}$ then $\cup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ contains at least $(n-t)(2n-4t)^{S-r-1}$ entries with value $\tilde{v}(q_1, \ldots, q_r)$, namely $(2n-4t)^{S-r-1}$ for each of at least $n-t$ choices of q_{r+1}. By Lemma 3, every entry of $W(q_1, \ldots, q_r, *, \ldots, *, p)$ is $\tilde{v}(q_1, \ldots, q_r)$.

(iii): If $q_r \in \mathit{Exposed}(r)$ then for every $q \in \mathit{Corr}$, $\tilde{v}(q_1, \ldots, q_r, q) = \perp_r$ (if $q_r \in \mathit{Fault}(r)$ this is explicit in the algorithm, and if $q_r \in \mathit{Fail}(r)$ then q_r sent no message to q in round r so $v(q_1, \ldots, q_r, q) = \perp_r$). By (ii) for $r+1$, if $q_r \in \mathit{Exposed}(r)$, $q_{r+1} \in \mathit{Corr}$ and $p \notin \mathit{Crash}$, then $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ consists of $(2n-4t)^{S-r-1}$ entries with value $\perp_r$. Hence if $q_r \in \mathit{Exposed}(r)$ and $p \notin \mathit{Crash}$ then $\cup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ contains at least $(n-t)(2n-4t)^{S-r-1}$ entries with value $\perp_r$, namely $(2n-4t)^{S-r-1}$ for each of at least $n-t$ choices of q_{r+1}. By Lemma 3, every entry of $W(q_1, \ldots, q_r, *, \ldots, *, p)$ is $\perp_r$.

(iv): We have that $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_0) = W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_1)$ unless $q_{r+1} \in \mathit{Exposed}(r+2) \setminus \mathit{Exposed}(r+1)$, by (ii) and (iii) for $r+1$. For the other l_{r+1} values of q_{r+1} we have by (iv) for $r+1$ that

$$\begin{aligned} &|\mathit{mult}(\tilde{v}(q_1, \ldots, q_r, q_{r+1}), W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_0)) \\ &\quad -\mathit{mult}(\tilde{v}(q_1, \ldots, q_r, q_{r+1}), W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_1))| \\ &\leq l_{r+2} \cdot l_{r+3} \cdots l_S. \end{aligned}$$

For each of these q_{r+1}, $\tilde{v}(q_1,\ldots,q_r,q_{r+1})$ is either $\tilde{v}(q_1,\ldots,q_r)$ or $\perp_r$, so we see

$$\begin{aligned}
&|mult(\tilde{v}(q_1,\ldots,q_r), \cup_{q_{r+1}=1}^{n} W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_0)) \\
&\quad -mult(\tilde{v}(q_1,\ldots,q_r), \cup_{q_{r+1}=1}^{n} W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_1))| \\
&+|mult(\perp_r, \cup_{q_{r+1}=1}^{n} W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_0)) \\
&\quad -mult(\perp_r, \cup_{q_{r+1}=1}^{n} W\,(q_1,\ldots,q_r,q_{r+1},*,\ldots,*,p_1))| \\
&\le l_{r+1}\cdot l_{r+2}\cdots l_S.
\end{aligned}$$

With this bound and the facts in (i) we can apply Lemma 4 to complete the proof.

Q.E.D.

Theorem 17 *The algorithm of this section has performance*

$$K \le \frac{\sup\{l_1 l_2\cdots l_S : l_1+\cdots+l_S \le t, \text{each } l_i \text{ a non-negative integer}\}}{(2n-4t)^{S-1}(2n-2t)}.$$

Proof: We have by (ii) and (iii) of Lemma 16 for $r = 1$ that $W(q_1,*,\ldots,*,p_0) = W(q_1,*,\ldots,*,p_1)$ unless $q_1 \in Exposed(2)\backslash Exposed(1)$. For these l_1 values of q_1 we have by (iv) for $r = 1$ that

$$|mult(v(q_1), W\,(q_1,*,\ldots,*,p_0)) - mult(v(q_1), W\,(q_1,*,\ldots,*,p_1))| \le l_2\cdot l_3\cdots + l_S$$

since $\tilde{v}(q_1) = v(q_1)$. We can apply Lemma 6 with $V = \cup_{q_1=1}^{n} W(q_1,*,\ldots,*,p_0)$, $W = \cup_{q_1=1}^{n} W(q_1,*,\ldots,*,p_1)$, $N = n(2n-4t)^{S-1}$, $m = l_1\cdots l_S$, $k = t(2n-4t)^{S-1}$ and $[a,b] = \rho(U)$ to prove that each of $w(p_0) = center_k(V)$ and $w(p_1) = center_k(W)$ lie in $\rho(U)$ and that

$$|w\,(p_0) - w\,(p_1)| \le \frac{l_1\cdots l_S}{(2n-4t)^{S-1}(2n-2t)}\cdot\delta\,(U)\,.$$

We finally note that as $l_1 = |Exposed\,(2)| - |Exposed(1)|$, $l_2 = |Exposed\,(3)| - |Exposed\,(2)|,\ldots,$ $l_S = |Exposed\,(S+1)| - |Exposed\,(S)|$, we have each l_i a non-negative integer and also $l_1 + l_2 + \ldots + l_S = |Exposed\,(S+1)| - |Exposed(1)| \le t$. This proves that our algorithm has, as claimed, performance

$$K \le \frac{\sup\{l_1 l_2\cdots l_S : l_1+\cdots l_S \le t, \text{each } l_i \text{ a non-negative integer}\}}{(2n-4t)^{S-1}(2n-2t)}.$$

The lower bound from §7 also applies to a synchronous failure-by-omission system, so that the algorithm of this section is asymptotically optimal.

9 Model of an Asynchronous System

In general, asynchronous systems seem to be harder to reason about than synchronous systems. The additional uncertainty in message delay leads to the possibility of race conditions, and many more possible sequences of activity need to be considered. In order to guard against error, we will use more detailed, formal descriptions of the system and the problem to be solved in the asynchronous case than we did for the synchronous systems earlier in this paper. These will be used to prove carefully the basic results about the power of the "adversary" against which the protocol must work. However, the discussion of the algorithm's correctness and performance, and the lower bound on any protocol, will be given using the same higher level modes of reasoning that were used for synchronous systems.

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.[2] 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[3]. We also assume that there are initial and decision states for every real number, not just for 0 and 1 as in [DDS].

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

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

[3]In many implementations, all messages will actually be received, and those that do not satisfy the guard will be buffered internally without affecting the computation, rather than being kept externally in the channel.

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 $\beta^p : Z^p \to \wp(C^{p,*} \times M)$ gives the set of messages being generated by processor p when it is in each state, together with the channel on which each message is to be sent. This function is required to satisfy the conditions $\beta^p(z) = \emptyset$ if $z \in Z^p_R$, and $|\beta^p(z)| \leq 1$ for all z. Thus no message is generated during a receiving step, and at most one message is generated in a sending step. The guard function $\gamma^p : Z^p \to \wp(C^{*,p})$ models the set of channels on which p attempts to receive messages when in each state. It is required to satisfy $\gamma^p(z) = \emptyset$ for $z \in Z^p_S$, modeling the fact that no messages may be received during a sending step. The state transition function $\delta^p : Z^p \times E^p \to Z^p$ indicates how the state of processor p changes when an event occurs at p. It 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[4] of messages for each channel. We write $\mathrm{st}(p, \kappa)$ specifying the state of processor p and $\mathrm{buff}(c, \kappa)$ 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 $c \in \gamma^p(\mathrm{st}(p, \kappa))$ and $m \in \mathrm{buff}(c, \kappa)$.

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 $\mathrm{st}(q, \kappa') = \mathrm{st}(q, \kappa)$ for $q \neq p$, $\mathrm{st}(p, \kappa') = \delta^p(\mathrm{st}(p, \kappa), e)$, $\mathrm{buff}(d, \kappa') = \mathrm{buff}(d, \kappa) \cup \{m' : (d, m') \in \beta^p(\mathrm{st}(p, \kappa))\}$ for $d \neq c$, and $\mathrm{buff}(c, \kappa') = (\mathrm{buff}(c, \kappa) - \{m\}) \cup \{m' : (c, m') \in \beta^p(\mathrm{st}(p, \kappa))\}$. If $e = \emptyset$ we define the failure-free result of (p, e) in κ to be the unique configuration κ' with $\mathrm{st}(q, \kappa') = \mathrm{st}(q, \kappa)$ for $q \neq p$, $\mathrm{st}(p, \kappa') = \delta^p(\mathrm{st}(p, \kappa), e)$ and $\mathrm{buff}(d, \kappa') = \mathrm{buff}(d, \kappa) \cup \{m' : (d, m') \in \beta^p(\mathrm{st}(p, \kappa))\}$ 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'

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

with $\mathrm{st}(q,\kappa') = \mathrm{st}(q,\kappa)$ for $q \neq p$, $\mathrm{st}(p,\kappa') = \delta^p(\mathrm{st}(p,\kappa),e)$, $\mathrm{buff}(d,\kappa') = \mathrm{buff}(d,\kappa)$ for $d \neq c$, and $\mathrm{buff}(c,\kappa') = (\mathrm{buff}(c,\kappa) - \{m\})$. If $e = \dagger$ or $e = \emptyset$ we define the failure result of (p,e) in κ to be the unique configuration κ' with $\mathrm{st}(q,\kappa') = \mathrm{st}(q,\kappa)$ for $q \neq p$, $\mathrm{st}(p,\kappa') = \delta^p(\mathrm{st}(p,\kappa),e)$ and $\mathrm{buff}(d,\kappa') = \mathrm{buff}(d,\kappa)$ 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 κ_1,(p_1,e_1),κ_2, (p_2,e_2),κ_3, …,(p_l,e_l),κ_{l+1} of configurations and events, starting and ending with a configuration, satisfying the conditions:

- for each processor p, $\mathrm{st}(p,\kappa_1) = z^p_{v,init}$ for some v;
- for each channel c, $\mathrm{buff}(c,\kappa_1) = \emptyset$;
- 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 $e_j = \dagger$.

An execution is an infinite alternating sequence κ_1,(p_1,e_1),κ_2,(p_2,e_2),κ_3,… whose prefixes of odd length are partial executions. In an execution or partial execution κ_1,(p_1,e_1),κ_2, (p_2,e_2),κ_3,…, 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 $m \in \mathrm{buff}(c,\kappa_i)$ 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 $c \in \gamma^{p_k}(\mathrm{st}(p_k,\kappa_k))$ and $e_k \neq \dagger$. 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 infinitely often. 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 $\mathrm{st}(p, \kappa_1)$, together with the subsequence of events (p_i, e_i) that involve p (i.e. for which $p_i = p$), form "the view of p in ρ". We note that $\mathrm{st}(p, \kappa_{l+1})$ is uniquely determined by the view of p in the partial execution $\kappa_1,(p_1, e_1),\kappa_2, (p_2, e_2),\kappa_3,\ldots,(p_l, e_l),\kappa_{l+1}$.

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 have the following useful extension result:

Lemma 18 *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 partial execution of a protocol. Then there is an admissible execution $\rho' = \kappa_1,(p_1, e_1),\kappa_2, (p_2, e_2),\kappa_3, \ldots$ of the protocol that has ρ as a prefix, such that the only processors that crash in ρ' are those that crashed in ρ, and the only processors that fail in ρ' are those that failed in ρ.*

Proof: We inductively construct (p_i, e_i) and κ_{i+1} for $i > l$. Let $p_i = (i \bmod n) + 1$. If p_i crashed in ρ, let $e_i = \dagger$ and $\kappa_{i+1} = \kappa_i$. Otherwise, consider the multiset of messages $\{m : m \in \mathrm{buff}(c, \kappa_i), c \in \gamma^{p_i}(\mathrm{st}(p_i, \kappa_i))\}$, 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 $m_0 \in \mathrm{buff}(c_0, \kappa_i)$. Now, let κ_{i+1} be the failure-free result of (p_i, e_i) in κ_i.

By construction, ρ' is an execution in which the only processors that crash are those that crash in ρ, and in which the only processors that fail are those that fail in ρ. 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 that captures the intuition about the behaviour of the "adversary" against which our algorithm needs to work.

Lemma 19 *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 $\tilde{\rho}$ with the same initial values, 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 $\mathrm{st}(p,\kappa_l)$ is a decision state for every processor $p \in R$. We will first construct an admissible, failure-free partial execution $\rho' = \kappa'_1,(p'_1,e'_1),\kappa'_2,(p'_2,e'_2),\ldots,\kappa'_l$, 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 $\mathrm{st}(p,\kappa'_i) = \mathrm{st}(p,\kappa_i)$ as long as p has not crashed in the first i steps of ρ. Now we can use Lemma 18 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 $\mathrm{st}(p,\kappa'_l) = \mathrm{st}(p,\kappa_l)$ is a decision state) we have thus completed the proof of the Lemma. Q.E.D.

If we use the terminology of knowledge (as used in [CM] and [HM] for example), we can explain this result as follows: it can never happen that a collection of processes, none of which has crashed, has knowledge that any failure has occurred. An immediate consequence is that no processor can ever know that a particular processor is faulty. Chandy and Misra have stated and proved a similar result (Theorem 8 of [CM]). Indeed, when only crash-failures are allowed (as in §5) the two results are equivalent. However, when omission failures are considered, our result is more powerful, as it shows that no collection of processors can know that a failure step has occurred, even when later messages have been received from the faulty processor. In contrast, in the model of [CM], a processor that has failed and then recovered knows that it failed and thus knows that it is among the faulty processors, and it can inform other processors of this fact. This awareness by a processor that it did omit to send a message, seems incompatible with the fundamental assumption of failure-by-omission systems, which is the assumption that faulty processors follow the normal protocol.

10 The Asynchronous Failure-by-Omission Model: The Algorithm

Each processor p acts according to the following algorithm, which we give first informally, as in [DLPSW] or the earlier sections of this paper:

- 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 "round r" of the algorithm, and we describe any message sent in round r as a round r message.

Now we give a formal account of this protocol using the model of §9, 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 begin$(c^{p,q,r}) = p$ and 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=$\emptyset$
  then return(w)
if z.mode=receive and e=$(c^{q,p,r}, m)$
  then begin
         w.valrec[q]=$m$
         if $|\{i : \text{w.valrec}[i] \neq \perp\}| < n-t$
           then return(w)
           else begin
```

```
            w.val ← $av_t(\{\text{w.valrec}[i] : \text{w.valrec}[i] \neq \perp\})$
            if w.round=$S$
              then begin
                   w.mode ←done
                   return(w)
                   end
              else begin
                   w.round ← w.round+1
                   for $i = 1, \ldots, n$ do w.valrec[$i$] ← $\perp$
                   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 20 *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 round r. (Thus if p has not crashed, it sends messages $\langle r, val_p^r\rangle$ in step (ii) of round r). Let U^r denote the multiset $\{val_p^r :\ p$ has not halted before executing step (iii) ofround $r\}$. 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 round 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) ofround 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 7 and 8, since $val_p^{r+1} = av_t(V_p^r)$. Q.E.D.

Theorem 21 *The algorithm above solves the t-resilient Approximate Agreement Problem using S rounds of communication, with performance*

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

Proof : The result follows immediately by applying the previous lemma S times. Q.E.D.

11 The Asynchronous Failure-by-Omission Model: The Lower Bound

The results of the previous section show that any performance can be obtained if enough communication is used. To prove a lower bound, we will consider only protocols that involve S rounds of message exchange. That is, we will assume that the protocols under discussion have the following form:

- For $r = 1, 2, \ldots, S$ successively
  - (i) Send some messages with r as first component to some processors.
  - (ii) Wait, trying to receive some subset of the messages with 1,2,. . .,r as the first component, until some condition on p's history is satisfied.

- Then decide on a value $w(p)$ that is some function of p's history. Afterwards, any pattern of sending and receiving messages is allowed, so long as no messages are sent that have a value less than or equal to S as their first component.

Thus each processor alternates between collections of sending states and collections of receiving states, with a collection of sending states and the following receiving states making up one round of the algorithm. Furthermore during the receiving states of round r, for $r \leq S$, the processor tries to receive only messages sent during the first r rounds. Each processor decides at the end of round S. Notice that the algorithm of the previous section has this form.

It will simplify our lower bound arguments if we consider processors that follow a particularly simple protocol of this sort, in which messages contain merely the history of the sender, each message is sent to everyone, and processors try to receive as much information as possible in each round. Thus we say that a ***full-information*** protocol[5] has the following form:

- For $r = 1, 2, \ldots, S$ successively
  - **(i)** Send a message $\langle r, \boldsymbol{hist} \rangle$ 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 any messages with $1,2,\ldots,r$ 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 $w(p)$ that 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 now give the intuitive reasons why this form of protocol is completely general, in that any S-round protocol can be implemented by a full-information protocol. The precise

[5]The full-information protocol we give here is a natural generalization to asynchronous systems of that described earlier for reasoning about synchronous algorithms.

meaning of this statement, and the formal proof, are given later. 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 a round r message to every processor. Similarly there is no harm in collecting as much information as possible in each round's receiving phase, and then ignoring whatever information is unwanted. To collect as much information as possible means to try to receive any possible message (from the current or previous rounds) and to continue doing so as long as the processor knows that more information will come. Thus each processor should wait in its receiving phase until it is possible that it has received everything that it will ever get, among the messages of rounds $1,\ldots,r-1$. 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. Any messages sent after a process has decided cannot be received until the recipient has decided, and so these messages cannot affect any decision value. Thus there is no need to send them.

Since full-information protocols have not previously been studied in asynchronous systems, we give here the proof that they can implement any t-resilient S-round protocol for solving a problem. We first observe that all S-round full-information protocols are identical except for the decision values associated to the decision states. In particular, all such protocols have exactly the same admissible executions, so that we can speak of an execution of a full-information protocol without specifying which decision function is to be used. Next we prove a lemma that relates failure-free executions of the full-information protocol to executions of a given protocol.

Lemma 22 *Let $\mathcal{A}$ be an S-round protocol such that in every admissible execution of $\mathcal{A}$, every processor that does not crash enters a deciding state. There is a construction that associates to any failure-free admissible execution σ of a full-information protocol a failure-free admissible execution* $\mathrm{full}(\sigma)$ *of $\mathcal{A}$, where the association has the property that the initial value of each processor is the same in* $\mathrm{full}(\sigma)$ *as in σ and that the decision value of p in* $\mathrm{full}(\sigma)$ *depends only on the view of p in σ.*

Proof: We will construct full(σ) inductively, round by round. Let the initial state of each processor in full(σ) be that with the same initial value as in σ. We thus have a partial execution where each processor is at the start of round 1, and the history of a processor in this depends only on the initial value of that processor in σ. As the induction step of the construction, assume that we have constructed a partial failure-free execution at the end of which each processor is at the start of round r, in such a way that the history of each processor in the partial execution depends only on the history of that processor up to the end of round $r-1$ in σ. We will first extend this partial execution to bring each processor to the end of the sending phase of round r. We do this by considering each processor in turn in round-robin order (i.e processor 1, then processor 2, ..., then processor n, then processor 1, etc.). If that processor has not yet reached the end of the sending phase of round r, extend the partial execution by a failure-free step $(p, \emptyset)$ of that processor. We show that eventually every processor will have reached the end of the sending phase of round r, and thus that the extensions constructed reach a limiting partial execution: if not, then there would be a prefix of the limiting extended (infinite, not necessarily admissible) execution in which every processor that is going to reach the end of the sending phase has done so, and there would be a processor p that will continue to send forever from its state at the end of that prefix. If we consider an admissible failure-free extension of that prefix, p would also never finish sending in that, and so p would not decide in that execution, contradicting the nature of $\mathcal{A}$.

Thus we see that we have extended the partial execution and brought every processor to the end of the sending phase of round r, and that the history of a processor in the extended failure-free partial execution depends only on its state at the end of round $r-1$, and thus only on the history of that processor up to the end of round $r-1$ in σ. We will now extend the partial execution further, to bring each processor to the end of round r. We first examine all the messages in transit in the configuration at the end of the partial execution constructed so far. We say that a message that was sent from processor p to processor q when p was in round r' of the partial execution (and that is still in transit) is *deliverable during round* r if in σ the round r' message from p to q was received by q during one of the rounds $r', r'+1, \ldots, r$. Once again consider each processor in turn, in round robin order. If that processor, say processor q, has reached the end of round r, do nothing. Otherwise, consider the set of messages that are both deliverable in round r and currently in transit on the channels on which processor q is currently trying to receive. If this set is empty, extend the partial execution with a step $(q, \emptyset)$, and otherwise extend the partial execution with a step $(q, (c, m))$, where m is the message among this set that was sent earliest, and c is the channel on which it was sent.

We show that eventually every processor will have reached the end of round r and thus that the extensions constructed reach a limiting partial execution: if not, there would be a prefix of the limiting execution in which every message that is going to be received has been received (since no messages are being added to the channels during the extensions). There would also be a processor p that would continue forever from its state in the configuration at the end of that prefix, trying to receive messages despite receiving nothing at each step. We see that in this infinite continuation p must never try to receive any of the messages that are still in transit in that configuration and that are deliverable in round r. We now consider a modified partial execution that has the same sequence of events as the prefix of full(σ) we constructed, but in which some configurations are the failure results of the preceding event, rather than the failure-free result. The events that are thus affected are exactly those in which (in full(σ)) a message was sent to p that was in transit at the start of the receiving phase of round r, but was not marked as being deliverable in round r. The modified partial execution is admissible, because the messages involved were sent by some set of at most t processors (as the corresponding messages in σ had not arrived at p when p finished round r of the full-information protocol.) The configuration at the end of the modified partial execution differs from that at the end of the prefix of full(σ) only in that the channels that end at p do not contain those messages that were sent to p, were still in transit at the start of the receiving phase of round r, and were not marked deliverable in round r. When the modified partial execution is extended to an execution in which no later failures occur, p will continue forever trying to receive messages despite receiving nothing at each step (as p will never try to receive the messages of rounds 1 through r that are in transit at the end of the modified partial execution, nor can it try to receive any messages added to the channels in later rounds.) Thus the extension of the modified partial execution is an admissible execution in which p does not crash, but does not reach a decision state.

This contradicts the nature of the protocol $\mathcal{A}$ and thus establishes the fact that eventually the construction of the prefix of full(σ) reaches a limit, in which every processor is at the end of round r.

If we consider the view of a processor p in the partial execution constructed, we see that it depends on the view of p at the end of round $r-1$, and on the messages in transit to p at that time that are deliverable in round r, but on nothing else. Now the processor's view at the end of round $r-1$ depends only on its view at the end of round $r-1$ in σ. Also we note that the view of p at the end of round r in σ enables us to determine the history of every other processor up to the end of the sending phase of the last round in which a message was sent that was delivered to p during the first r rounds of σ, and this in turn allows us to deduce the views of those processors to the corresponding point in the partial execution constructed. The latter however is exactly what is needed to determine the messages that are deliverable to p in round r. This completes the establishment of the induction step of the construction.

Thus we can construct a partial execution to be a prefix of full(σ), so that at the end of this partial execution each processor has reached the end of round S of $\mathcal{A}$, and therefore has decided on a value that depends only on the processors view up to the end of round S of σ. This failure-free partial execution can be extended to a failure-free admissible execution full(σ), which is exactly what is required by the lemma. Q.E.D.

Theorem 23 *Let $\mathcal{A}$ be an S-round protocol such that in every admissible execution of $\mathcal{A}$, every processor that does not crash enters a deciding state. Then there exists an S-round full-information protocol $\mathcal{P}$ satisfying the following condition: for every admissible execution ρ of $\mathcal{P}$ there is an admissible execution ρ' of $\mathcal{A}$, where the initial value of each processor is the same in ρ' as in ρ, the same processors crash in ρ' as in ρ, and the decision value of each processor that does not crash is the same in ρ' as in ρ.*

Proof: To construct $\mathcal{P}$, we need only specify the decision value that a processor p should choose after the completion of round S, given the history h of p up to that point. First, we choose some failure-free admissible execution ξ of the full-information protocol such that p has history h in ξ up to the end of round S. Note that the history of p in ξ is simply h

followed by an infinite number of steps in which no message is received. We now consider the execution full(ξ) of $\mathcal{A}$, and let the decision value of p in $\mathcal{P}$ after h be chosen to be the decision value of p in $\mathcal{A}$ in full(ξ). That such a value exists follows from the assumption that processors always decide in $\mathcal{A}$ if they do not crash, and the fact that this value is independent of the choice of ξ is immediate from the property claimed for the association between ξ and full(ξ). Thus we have defined $\mathcal{P}$.

Now, take any admissible execution ρ of $\mathcal{P}$, and let R denote the set of processors that do not crash during ρ. By Lemma 19, there is an admissible, failure-free execution $\tilde{\rho}$ of $\mathcal{P}$ with the same initial values and such that every processor in R has the same view in $\tilde{\rho}$ as it has in ρ. Now full($\tilde{\rho}$) is a failure-free admissible execution of $\mathcal{A}$, and thus in it every processor enters a decision state. We choose a partial execution of $\mathcal{A}$ that is a prefix of full($\tilde{\rho}$) and in which every processor has decided. We then extend this partial execution by a crash event $(q, \dagger)$ for each processor $q \notin R$, and then use Lemma 18 to extend the resulting partial execution of $\mathcal{A}$ to an admissible (infinite) execution ρ' of $\mathcal{A}$, in which no additional processors crash.

The initial value of a processor in ρ' is the same as the initial value in full($\tilde{\rho}$), which is the same as its initial value in $\tilde{\rho}$, which is the same as that in ρ. We observe that by construction the processors that crash in ρ' are exactly those that are not in R, that is, those that crashed in ρ. For each processor $p \in R$, $\tilde{\rho}$ is one of the possible choices of ξ that could have been used to compute the decision value of p in $\mathcal{P}$, and thus we see that the decision value of p in full($\tilde{\rho}$) is the same as the decision value of p in ρ. Since ρ' is an extension of a prefix of full($\tilde{\rho}$) and p has already entered a decision state in that prefix, the decision value of p in ρ' is the same as in full($\tilde{\rho}$), and thus is the same as the decision value of p in ρ. Q.E.D.

To say that a protocol $\mathcal{P}$ 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 24 *For any S-round protocol $\mathcal{P}$ that solves the t-resilient Approximate Agreement Problem in an asynchronous, 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 that 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 $\nu = \lceil \frac{n-t}{t} \rceil$. In the following discussion we will assume that $\mathcal{P}$ is a full-information protocol. If $\mathcal{P}$ is some other S-round protocol, we can find a full-information protocol to implement it, use the proof below to find an admissible execution of that, and then take the corresponding execution of the original protocol. This will have the desired properties.

We will use induction on S. We first prove the theorem for $S = 1$.

We denote by p_α ($\alpha = 0, \ldots, \nu - 1$) the processor $(\alpha + 1)t + 1$, and by p_ν processor 1. Now we will describe a chain of admissible executions ρ_0, $\rho_1, \ldots, \rho_{\nu+1}$ such that processor p_α has the same view in executions ρ_α and $\rho_{\alpha+1}$, 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_0 in that execution must be 1. Similarly the decision value of p_ν in $\rho_{\nu+1}$ must be 0. From these facts it follows by a standard argument (as in §5 and 7) that for some α the processes $p_{\alpha-1}$ and p_α 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 $t + 1, \ldots, n$ before entering its decision state. For $\alpha = 1, \ldots, \nu - 1$ the execution ρ_α has processors $1, 2, \ldots, \alpha t$ with initial value 0, and processors $\alpha t + 1, \ldots, n$ with initial value 1. No processor crashes or omits to send, and $p_{\alpha-1}$ enters its decision state after receiving round 1 messages from processors $1, \ldots, (\alpha - 1)t$ and $\alpha t + 1, \ldots, n$, while every other processor (in particular p_α) receives round 1 messages from processors $1, \ldots, \alpha t$ and $(\alpha + 1)t + 1, \ldots, n$ before entering its decision state. The execution ρ_ν has processors $1, 2, \ldots, \nu t$ with initial value 0, and processors $\nu t + 1, \ldots, n$ with initial value 1. No processor crashes or omits to send, and $p_{\nu-1}$ enters its decision state after receiving round 1 messages from processors $1, \ldots, (\nu - 1)t$ and $\nu t + 1, \ldots, n$, while every other processor (in particular p_ν) receives round 1 messages from processors $1,\ldots,n-t$ 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 $1,\ldots,n-t$.

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 will construct an $(S-1)$-round full-information protocol $\mathcal{Q}$. To describe $\mathcal{Q}$ we have to specify the decision value chosen by p at the end of round $S-1$ after a given history h. First, we choose some failure-free admissible execution ξ of $\mathcal{Q}$, such that p has history h in ξ up to the end of round $S-1$. In the next paragraph we will show the construction of an execution $\mathrm{ext}^p(\xi)$ of the S-round full-information protocol $\mathcal{P}$. The construction will have the property that the history of p in $\mathrm{ext}^p(\xi)$ will depend only on the history h, and not on the choice of ξ. Now let the decision value of p in $\mathcal{Q}$ after history h be the decision value of p in the execution $\mathrm{ext}^p(\xi)$ of $\mathcal{P}$. Because this depends only on h, $\mathcal{Q}$ is well-defined. We can think of processor p as imagining that it is running protocol $\mathcal{P}$, and using the actual execution of $S-1$ rounds of message exchange to guide it during an imagined execution of the S-round protocol $\mathcal{P}$ that will determine its decision value.

The execution $\mathrm{ext}^p(\xi)$ is identical to ξ during the first $S-2$ rounds. Thus we only need to describe the final two rounds of message exchange.[6] During the sending phase of round $S-1$ every processor sends its history to every processor, as required by the full-information protocol. Let R_p denote the set of $n-t$ processors such that p receives every message from each processor in R_p during ξ (such a set exists since p completed round $S-1$.) During the receiving phase of round $S-1$ of $\mathrm{ext}^p(\xi)$, each processor p' receives those messages of rounds $1,\ldots,S-1$ that are sent from processors in R_p and that are not received by p' in earlier rounds, and no other messages. This brings each processor to the end of round $S-1$. In the sending phase of round S, each processor sends its history to every processor. In the

[6] 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 §9, we would also need to specify the order in which processors take steps and the order in which the messages are received during each round. Any consistently applied choice would be suitable, for example, allowing processors to take steps in round robin order, and having a round i_1 message from processor p_1 received before a round i_2 message from processor p_2 if $p_1 < p_2$ or $(p_1 = p_2$ and $i_1 < i_2)$.

receiving phase of round S, each processor receives the round S message from the processors in R_p, and no other messages. This brings every processor to the end of round S, as each has received all the messages from the processors in R_p. This completes the description of $\mathrm{ext}^p(\xi)$.

The induction hypothesis 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' that 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 19 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 view in ρ_α and $\rho_{\alpha+1}$. Furthermore, the view of $p_0 = p'$ in execution ρ_1 is the same as the view of p' during the execution $\mathrm{ext}^{p'}(\rho')$, a possible execution of $\mathcal{P}$ imagined by p' to determine its decision value during execution ρ' of protocol $\mathcal{Q}$, and so during ρ_1, p_0 must decide on value $w(p')$. Similarly the view of $p_\nu = q'$ during execution ρ_ν will be the same as the view of q' during $\mathrm{ext}^{q'}(\rho')$, 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

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

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 rounds $1, \ldots, S-2$. We next describe rounds $S-1$ and S of the executions ρ_α.[7] In the sending phase of round

[7]In the formal model of §9, we would also need to specify in which order the processors take steps, and in which order the various messages in a round are received, in order to completely specify an 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 $\mathcal{Q}$.

$S-1$ during each execution ρ_α, every processor sends its history to every processor. 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 are received by p in rounds $1,\ldots,S-2$. (Recall that $R_{p'}$ denotes the set of $n-t$ processors used in $\mathcal{Q}$ by processor p' to determine its decision value). Thus $M_{p'}(p)$ is the set of messages that p receives in round $S-1$ of the execution $\mathrm{ext}^{p'}(\rho')$ of $\mathcal{P}$ that p' imagines when computing its decision at the end of execution ρ' of $\mathcal{Q}$. 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 are not received by p in rounds $1,\ldots,S-2$. 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 the receiving phase of round $S-1$ of the execution ρ_α for $\alpha=1,\ldots,\nu$ by requiring that the set of messages received by processor p during round $S-1$ be $M_{p'}(p)$ if $(\alpha-1)t<p\leq n-t$, and be $M_{q'}(p)$ otherwise.

In the sending phase of round S during each execution ρ_α, each processor sends its history to every processor. The construction of the receiving phase of round S of the executions will be given separately for the cases $\nu\geq 2$ and $\nu=1$. First suppose $\nu\geq 2$. In the execution ρ_1, during round S, processor p_0 receives all the messages sent by processors $1,\ldots,n-t$ during rounds $1,\ldots,S$ that it had not previously 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 rounds $1,\ldots,S$, that it has 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 have 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 have not received before. The execution ρ_ν has round 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 round 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.

12 The Asynchronous 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 §9, we say that an execution or partial execution is a ***crash-failure*** execution or partial execution if every configuration is the failure-free result of the previous step unless the previous event 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 protocol for the failure-by-omission model, and then applying Theorem 24. Thus the lower bound of §11 also applies to the crash-failure model, and so the protocol of §10 remains optimal in the more restricted crash-failure system.

We introduce full-information protocols for crash-failure systems, with the following form:

- For $r = 1, 2, \ldots, S$ successively

  (i) Send a message $\langle r, \textit{hist} \rangle$ to each processor (including p itself) where $\textit{hist}$ 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 $1,2,\ldots,r$ 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 $w(p)$ that is some function of p's history, and thereafter do not send or try to receive messages.

Any S-round protocol, such that in every admissible crash-failure execution each processor that does not crash reaches a decision, can be implemented by a crash-failure full-information protocol. The proof of this fact is follows exactly the same method as the proof of Theorem 23.

We now show that any crash-failure full-information protocol can be simulated by a full-information protocol for the failure-by-omission model.

Lemma 25 *Let $\mathcal{P}$ be an S-round crash-failure full-information protocol. Then there exists an S-round failure-by-omission full-information protocol $\mathcal{Q}$, such that for every admissible execution ρ of $\mathcal{Q}$, there is an admissible crash-failure execution ρ' of $\mathcal{P}$, where the initial value of each processor is the same in ρ' as in ρ, the same processors crash in ρ' as in ρ, and the decision value of each processor that does not crash is the same in ρ' as in ρ.*

Proof: We note that different S-round crash-failure full-information protocols differ only in the way decision values are assigned to decision states, and so we can talk about an execution of the S-round crash-failure full-information protocol without needing to first specify which such protocol is involved. Thus we will first describe a construction that associates to any admissible failure-free execution σ of an S-round failure-by-omission full-information protocol an admissible failure-free execution convert(σ) of the crash-failure full-information protocol, where the association has the property that the initial value of each processor is the same in convert(σ) as in σ, and the view of each processor in convert(σ) depends only on that processor's view in σ.

We will construct convert(σ) inductively, round by round. Let the initial state of each processor in convert(σ) be that with the same initial value as in σ. We thus have a partial execution where each processor is at the start of round 1, and the history of a processor in this depends only on the initial value of that processor in σ. As the induction step of the construction, assume that we have constructed a partial execution at the end of which each processor is at the start of round r, in such a way that the history of each processor in the partial execution depends only on the history of that processor up to the end of round $r-1$ in σ. We will first extend this partial execution to bring each processor to the end of the sending phase of round r. We do this by considering each processor in turn in order (i.e processor 1, then processor 2, ..., then processor n), and extending the partial execution by having that processor send its history to every processor. Thus we see that we have extended the partial execution and brought every processor to the end of the sending phase of round r, and that the history of a processor in the extended failure-free partial execution depends only on its state at the end of round $r-1$, and thus only on the history of that processor up to the end of round $r-1$ in σ.

We will now extend the partial execution further, to bring each processor to the end of round r. We first examine all the messages in transit in the configuration at the end of the partial execution constructed so far. We say that a round r' message that was sent from processor p to processor q (and that is still in transit) is *primary for round r* if in σ the round r' message from p to q was received by q during round r. Once again consider each processor in turn. When considering processor p, we will describe three successive extensions to the partial execution constructed so far. First, extend the partial execution by steps in which p receives all the messages in transit to it that are primary for round r. Next, if the round r message from p to itself was not primary for round r, further extend the partial execution by a step in which p receives the round r message from itself. This ensures that there is some set of $n-t$ processors including p itself such that p has received r messages from each of these processes during the partial execution constructed so far. Finally, consider all the messages that were in transit to p and primary for round r, together with all the messages whose whose receipt is recorded in the histories that are the contents of messages in transit to p that were primary for round r. Foe each such message, say a round r' message from p' to q', extend the partial execution by the receipt of all messages sent by p' to p in rounds $1, \ldots, r' - 1$ that have not yet been received. Notice that the contents of these messages can be deduced from the contents of the messages received by p that are primary for round r. This brings processor p to the end of round r. If we consider the view of a processor p in the partial execution constructed, we see that it depends on the view of p at the end of round $r - 1$, and on the messages in transit to p that are primary for round r, but on nothing else. Each of these can be deduced from the view of p at the end of round r in σ. This completes the establishment of the induction step of the construction.

Now, to construct $\mathcal{Q}$, we need only specify the decision value that a processor p should choose after the completion of round S, given the history h of p up to that point. First, we choose some admissible failure-free execution ξ of the full-information protocol such that p has history h in ξ up to the end of round S. Note that the history of p in ξ is simply h followed by an infinite number of steps in which no message is received. We now consider the execution convert(ξ) of $\mathcal{P}$, and let the decision value of p in $\mathcal{Q}$ after h be chosen to be the decision value of p in $\mathcal{P}$ in convert(ξ). The fact that this value is independent of the choice of ξ is immediate from the property claimed for the association between ξ and convert(ξ). Thus we have defined $\mathcal{Q}$. Now given the admissible execution ρ of $\mathcal{Q}$, let σ be an admissible failure-free execution of $\mathcal{Q}$ in which each processor that does not crash in ρ has the same view as in ρ. Let ρ' be defined to be the same as convert(σ) except that every processor that crashes in ρ crashes in rho' exactly at the end of round S. That is, each step $(p.\emptyset)$ for such a processor after the end of round S, is replaced by a step $(p, \dagger)$. With this choice of ρ' we see that all the conditions required in the lemma are satisfied. Q.E.D.

Finally we complete our analysis by showing that the lower bound proved in §11 applies to S-round protocols for solving Approximate Agreement in crash-failure systems.

Theorem 26 *For any S-round protocol $\mathcal{P}$ that solves the 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 that 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: In the following discussion we will assume that $\mathcal{P}$ is a full-information protocol. If $\mathcal{P}$ is some other S-round protocol, we can find a full-information protocol to implement it, use the proof below to find an admissible execution of that, and then take the corresponding execution of the original protocol. This will have the desired properties.

By Lemma 25, we can find an S-round failure-by-omission full-information protocol $\mathcal{Q}$that implements protocol $\mathcal{P}$. We first show that $\mathcal{Q}$ solves the Approximate Agreement problem. Let ρ be any admissible execution of the protocol $\mathcal{Q}$. The corresponding crash-failure execution ρ' of $\mathcal{P}$ is admissible, and since $\mathcal{P}$ solves the crash-failure Approximate Agreement problem, we must have that the decision value of each processor that does not crash in ρ' lies within the range of the initial values. Since the initial values, the set of processors that do not crash, and their decision values, are all the same in ρ' as in ρ, we deduce that the decision value of each processor that does not crash in ρ lies within the range of the initial values.

Theorem 24 implies 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' of $\mathcal{P}$ therefore has the properties required for this theorem.

Q.E.D.

13 Summary of Results

For synchronous systems, we have presented new algorithms to solve S-round t-resilient Approximate Agreement in each of the crash-failure, failure-by-omission and Byzantine failure models. The algorithms have the nice property that they can be easily modified so that they can be started without knowing how many rounds of communication will be used, and they give intermediate results that get closer and closer together, reaching exact agreement once $S \geq t + 1$.

For fixed S, the algorithms' performance is asymptotic to the best possible as $n/t \rightarrow \infty$. Finding exactly matching lower bounds and upper bounds remains an open question. The results in this paper for synchronous systems are summarized in the following table (all except the lower bound for the Byzantine model are new results)

Model	Algorithm	Lower Bound
Crash-Failure	$K \leq \frac{L(S)}{(2n-2t)^S}$	$K \geq \frac{L(S)}{(2n+3t)^S}$
Failure-by-Omission	$K \leq \frac{L(S)}{(2n-2t)(2n-4t)^{S-1}}$	$K \geq \frac{L(S)}{(2n+3t)^S}$
Byzantine Failure	$K \leq \frac{L(S)}{(n-2t)(n-4t)^{S-1}}$	$K \geq \frac{L(S)}{(n+t)^S}$

where $L(S) = \sup(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t,$ each l_i a nonnegative integer).

All the algorithms given for synchronous systems are similar in that they involve processors exchanging information and then forming multisets $W(q_1, \ldots, q_r, *, \ldots, *, p)$ to represent all the information $\{v(q_1, \ldots, q_r, q_{r+1}, \ldots, q_S, p) : q_j = 1, 2, \ldots, n \text{ for } j > r\}$, using the operations of combining multisets and removing extreme values repeatedly to increase the amount of unanimity in each multiset. In the failure-by-omission and Byzantine failure models we also try to detect which processors are faulty, and ignore messages sent by processors that have been detected.

The algorithms introduced here for synchronous systems require exponential amounts ($O(n^{S+1})$ bits) of message traffic[8], like most other consensus or Byzantine Agreement algorithms. In the crash-failure model it is easy to modify our algorithm to use only $O(n^3S)$ bits of message traffic since a processor's history (and thus its message) is determined by (and so can be encoded by) what it knows of the n initial values and the last message known to have been sent by each other processor. Similarly in the omission-failure model we can modify our algorithm to use only $O(n^4S^2)$ bits of message traffic, since a processor's view of the system is determined by what it knows of the n initial values and the list of which of the n^2 messages in each prior round it knows were sent. A detailed analysis of the complexity of determining a processor's history in the crash-failure and failure-by-omission models can be found in [MT]. In the Byzantine model Coan has introduced a transformation that can encode algorithms of our type so as to require only polynomial communication ([C]). However Coan's transformation costs a few rounds of communication, and so the transformed

[8]assuming a fixed precision for real numbers

algorithm will not have performance that is asymptotic to optimal. The decision in practice between Coan's transformation of our algorithm, and the iteration of the one round algorithm of [DLPSW] (which involves only $O(n^2S)$ message traffic) will depend on the details of the system.

In asynchronous crash-failure or failure-by-omission systems, we have introduced a simple round-by-round algorithm that solves the t-resilient Approximate Agreement problem using S rounds of communication. This algorithm has performance $K \leq \lceil \frac{n-t}{t} \rceil^{-S}$, which is optimal in either model, by a new lower bound that depends on a lemma expressing the intuition that the adversary can do no more harm by causing failures than it could merely by delaying messages.

References

[CM] K. Chandy, J. Misra, "How Processes Learn", Distributed Computing, 1, 40–52 (1986).

[DDS] D. Dolev, C. Dwork, L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus", JACM, 34,1, 77–97 (1987).

[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).

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

[FL] M. Fischer, N. Lynch, "A Lower Bound for the Time to Assure Interactive Consistency", Information Processing Letters 14, 4, 183–186 (1982).

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

[HM] J. Halpern, Y. Moses, "Knowledge and Common Knowledge in a Distributed Environment", Proceedings of the 3rd ACM Symposium on Principles of Distributed

Computing, 50–61, August 1984. A revised version will appear as IBM Research Report RJ 4421, August 1987.

[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).

[LaM] 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.

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