Distributed Computing (1990) 4:9–29

DISTRIBUTED COMPUTING

Asymptotically optimal algorithms for approximate agreement*

A.D. Fekete
Department of Computer Science, University of Sydney, NSW 2006 Australia

Alan Fekete was born in Sydney Australia in 1959. He studied Pure Mathematics and Computer Science at the University of Sydney, obtaining a B.Sc.(Hons) in 1982. He then moved to Cambridge, Massachusetts, where he obtained a distributed Ph.D. degree, awarded by Harvard University's Mathematics department for work supervised by Nancy Lynch in M.I.T.'s Laboratory for Computer Science. He spend the year 1987–1988 at M.I.T. as a postdoctoral Research Associate, and is now Lecturer in Computer Science at the University of Sydney. His research concentrates on understanding the modularity in distributed algorithms, especially those used for concurrency control in distributed databases.

Abstract. This paper introduces some algorithms to solve crash-failure, failure-by-omission and Byzantine failure versions of the Byzantine Generals or consensus problem, where non-faulty processors need only arrive at values that are close together rather than identical. For each failure model and each value of S, we give a t-resilient algorithm using S rounds of communication. If $S=t+1$, exact agreement is obtained. In the algorithms for the failure-by-omission and Byzantine failure models, each processor attempts to identify the faulty processors and corrects values transmitted by them to reduce the amount of disagreement. We also prove lower bounds for each model, to show that each of our algorithms has a convergence rate that is asymptotic to the best possible in that model as the number of processors increases.

Key words: Distributed algorithms – Fault-tolerant consensus problems – Byzantine agreement problem – Approximate agreement

* A preliminary version of this paper has appeared in the Proceedings of the 5th ACM Symposium on Principles of Distributed Computing (August 1986). This work was begun in the Department of Mathematics, Harvard University, and completed at the Laboratory for Computer Science at Massachusetts Institute of Technology. The work was supported in part (through Professor N. Lynch) 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

1 The problem and statement of results

An important question in the design of fault-tolerant distributed systems is how to enable non-faulty communicating processors to agree even when faulty processors in the system are interfering by providing different correct processors with different information. Examples of applications include agreeing on whether to commit a database transaction and agreeing on which copy of a file is the primary copy. Classical formulations of this problem are known as the interactive consistency problem and the Byzantine Generals problem [6]. These problems have been studied in several models of computation. In the most general synchronous model it has been found [14, 11, 7] that any solution resilient to t faulty processors requires at least $3t+1$ processors and at least $t+1$ rounds of communication in the worst case. The bound on rounds of communication also holds in more restrictive failure models. In some practical situations complete agreement is not required – e.g. in synchronizing clocks [8, 10, 9] or reading a sensor, it is often good enough if all the values held by different processors are close together. We may hope for protocols using fewer rounds of communication for this problem called the approximate agreement problem, which was first studied in [4].

In this paper we study a t-resilient approximate agreement problem in this form: there are n proces-

sors labeled $1, 2, \ldots, n$. These processors are linked by a complete, synchronous, fault-free, point-to-point network which is the only means of interprocess communication. 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. We 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 $Fault = \{1, 2, \ldots, n\} \setminus Corr$ and set $f = |Fault|$. We also let *Crash* denote the subset of *Fault* consisting of processors that halted during the execution. Each processor p has an initial value $v(p)$ which is a real number and at the end of any execution of the algorithm for which $f \leq t$ each correct processor p must arrive at a new value $w(p)$ satisfying a validity cndition: in the crash-failure and failure-by-omission models this is that for correct p, $w(p)$ must lie within the range of the initial values. In the Byzantine model we do not trust the initial values of faulty processors, so we insist that for correct $p, w(p)$ must lie within the range of the initial values of the correct processors. Naturally we put no requirement on the final state of the faulty processors, nor 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 $\hat{U} = \{v(p): p \in Corr\}$. We can express the validity condition in the failure-by-omission and crash-failure models by "if $|Fault| \leq t$ and $p \in Corr$ then $w(p) \in \rho(U)$". Similarly in the Byzantine model the validity condition is "if $|Fault| \leq t$ and $p \in Corr$ then $w(p) \in \rho(\hat{U})$".

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

$$K = \sup \frac{\delta(\{w(p): p \in Corr\})}{\delta(U)}$$

and in the Byzantine model by

$$K = \sup \frac{\delta(\{w(p): p \in Corr\})}{\delta(\hat{U})},$$

in each case the supremum being taken over all executions with $|Fault| \leq t$ (so a good algorithm is one with a low value for K). Notice that the identification of processors as faulty or correct is available to us when analyzing the execution, but is not necessarily known to the processors during the running of the algorithm.

Since the failure-by-omission model allows any faulty behavior that is allowed in the crash-failure model (as well as other forms of faulty behavior), we see that any alogirthm that solves the approximate agreement problem in the failure-by-omission model also solves it in the crash-failure model, and the performance K of the algorithm in the crash-failure model is less than or equal to the performance in the failure-by-omission model. Similarly any algorithm that solves the approximate agreement problem in the Byzantine failure model also solves it (with a performance at least as good) in the failure-by-omission and crash-failure models.

For the Byzantine model, [4] gives an algorithm using only one round of communication, valid when $n > 3t$, with performance $K = \lceil (n-2t)/t \rceil^{-1}$. 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 $K = (\lceil (n-2t)/t \rceil)^{-S}$. The paper [4] raises the question of whether it is possible to design an S-round algorithm that combines the information from different rounds to obtain better performance. This paper addresses this issue and also considers the impact of changing the failure model.

First we give a new one round algorithm valid in the crash-failure model whenever $n > t$. This algorithm is very similar to the algorithm of [4], but uses a different method of removing extreme values and averaging, specially tailored to the more restricted failure model. Our algorithm attains a performance $K = t/(2n-2t)$, which is asymptotically (as $n/t \longrightarrow \infty$) twice as good as the algorithm of [4], since $\lim_{n/t \to \infty} \frac{t/(2n-2t)}{\lceil (n-2t)/t \rceil^{-1}} = \frac{1}{2}$. This im-

proved performance may be understood through the following intuitive reasoning: a Byzantine fault gives different processors two different ideas, say v and w, of the initial value of a processor. Two crash failures can produce a similar effect, by giving one processor the impression that a pair of initial values are v and no value, while another processor receives the impression that the same initial values are no value and w. Thus one can expect an algorithm tolerating t crash failures to have similar performance to one that tolerates $t/2$ Byzantine faults.

If we iterate this one-round algorithm for S rounds, we might expect to obtain a performance of $(t/(2n-2t))^S$. In fact we show that we actually do much better – the iterated algorithm has performance

$$K \leq \frac{\sup(l_1 \ldots l_S : l_1 + \ldots + l_S \leq t, \text{ each } l_i \text{ a nonnegtive integer})}{(2n-2t)^S}.$$

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

$$K \leq \frac{t^S}{S^S(2n-2t)^S}.$$

The reason the iterated algorithm does so well is that differences between processors' values at the end of a round are entirely due to failures that occur during that round – a processor that fails earlier provides the same lack of information to all others, and one that fails later provides the same good information to all the processors in the round in question. Thus if l_r processors crash during round r, then the span of values is multiplied by a factor $l_r/(2n-2t)$ during that round. Since each faulty processor can crash during at most one round, we see that $l_1 + l_2 + \ldots + l_S \leq t$.

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

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

for any algorithm using S rounds of communication, whenever $n > t+1$. Thus for fixed S our iterated algorithm for the crash-failure model is asymptotic to the optimum (that is, the ratio of our algorithm's performance to the lower bound tends to the limit 1) as n/t increases, that is, as the number of processors increases relative to the number faulty.

In the failure-by-omission and Byzantine failure models we cannot say that a processor sends different information to different processors during only one round. Thus we introduce algorithms which use *failure detection*. We provide a way for one processor to detect that another processor has failed. Each processor ignores any values sent to it by a processor known to have failed, so that after a processor is universally detected as faulty it will be unable to provide different information to different processors. We also use operators that remove extreme values from collections of values in such a way that they eliminate any differences introduced between processors' information by a faulty processor that escapes universal detection. Thus each faulty processor can cause differences between other processors' information only once, namely in the round in which it becomes universally detected as faulty. A technique of detection of faulty processors was used earlier in the similar problem of inexact agreement (where there is an a priori bound on the diameter of initial values of correct processors) in [12].

We give an algorithm using failure detection in the failure-by-omission model. This algorithm is valid when $n > 2t$, and has performance

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

which is asymptotic to optimal (in the sense that for fixed S the ratio of the algorithm's performance to the lower bound tends to the limit 1 as n/t tends to infinity), since the lower bound for the crash-failure model applies also to the more general failure-by-omission model.

For the Byzantine failure model, we offer an S-round algorithm valid when $n > 4t$, with performance

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

For large n this is asymptotic to S^S times better than the performance of the iterated algorithm from [4]. We prove the lower bound for an S-round algorithm resilient to t Byzantine failures:

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

whose asymptotic form is due to [4]. Therefore our algorithm is asymptotic to the best possible (in the sense that for fixed S the ratio of the algorithm's performance to the lower bound tends to the limit 1 as n/t tends to infinity).

It is worth noting that if $S = t+1$ the expression $\sup(l_1 \ldots l_S : l_1 + \ldots + l_S \leq t$, each l_i a nonnegative integer) is zero, as one of the l_i must be zero, and so each of our algorithms (when run for $t+1$ rounds) produces exact agreement. Even in the By-

zantine failure and failure-by-omission models (where our algorithms are not iterated applications of a one round algorithm) our algorithm for S rounds starts by doing all the communication of the $S-1$ round algorithm, so our algorithm can be modified to output tentative values (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 once $t+1$ rounds have passed.

The multi-round algorithms introduced here for the failure-by-omission and Byzantine failure models require exponential amounts ($O(n^{S+1})$ bits) of message traffic[1], like many other consensus or Byzantine Agreement algorithms. In the failure-by-omission model we can modify our algorithm to use only $O(n^4 S^2)$ bits of message traffic, since a processor's view of the system (and thus its message) is determined by (and so can be encoded 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 [13]. In the Byzantine model Coan [1] has introduced a transformation which can encode algorithms of our type so as to require only polynomial communication. However Coan's transformation costs a few rounds of communication, and so the transformed 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 [4] (which involves only $O(n^2 S)$ message traffic, but produces values that are farther apart) will depend on the details of the system.

The reader should note that our algorithms are optimal only asymptotically, as the number of processors n increases relative to the number of faults tolerated t. For systems in which n is not very large relative to t (say, $n \approx 50t$) the performance of our algorithms is far from optimal, although our algorithms still give closer final values than merely iterating the algorithm of [4]. Finding algorithms that are precisely optimal in each failure model remains an open question.

In § 2 we give the notation and technical lemmas we will use later. § 3 describes the iterated algorithm for the crash-failure model, and analyses it. Next § 4 describes the multi-round algorithms for the failure-by-omission and Byzantine failure models. In § 5 and § 6 these algorithms' correctness and performance are analyzed. In § 7 and § 8 we prove the lower bound for the Byzantine failure model and the crash-failure model, respectively. Finally in § 9 we summarize our results.

[1] Here we assume a fixed precision for real numbers

2 Notation and lemmas

In order to give the algorithms precisely, we introduce the language of multisets. A formal account appears in [4] but for our purposes it is enough to think of a multiset as an unordered collection of entries, with values that need not be distinct. For each value v and multiset V we denote the number of entries in V with value v (the *multiplicity* of v) by $mult(v, V)$. The values may be either real numbers or the special symbols $\perp_r$ (which we will use to denote a value not received in round r because, for example, a processor failed to send it). We define union, intersection, cardinality, sum and mean for multisets in the obvious ways, so for any v, $mult(v, V \cap W) = min(mult(v, V), mult(v, W))$ and $mult(v, V \cup W) = mult(v, V) + mult(v, W)$, and also $|V| = \sum_v mult(v, V)$, $\sum V = \sum_v v \cdot mult(v, V)$ and $mean(V) = |V|^{-1} \sum V$. Also let $double(V)$ be defined by $mult(v, double(V)) = 2 \cdot mult(v, V)$.

As in [4] 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 failure-by-omission model 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.

As examples:

- $\{-1,-1,0\}\cup\{0,1\}=\{-1,-1,0,0,1\}$
- $\{-1,-1,\perp_1\}\cup\{0,\perp_1,\perp_2\}$ $=\{-1,-1,0,\perp_1,\perp_1,\perp_2\}$
- $\{-1,-1,0,0\}\cap\{-1,0,0,1\}=\{-1,0,0\}$
- $|\{-1,-1,0\}|=3$
- $|\{-1,-1,0,\perp_1\}|=4$
- $red_2(\{-1,-1,-1,0,0,1\})=\{-1,0\}$
- $red_1(\{-1,-1,0,\perp_1,\perp_2\})=\{-1,0,\perp_1\}$
- $chop_1^2(\{-1,0,0,\perp_2,\perp_2\})$ $=\{-1,-1,0,0,0,0,\perp_2,\perp_2\}$
- $chop_3^2(\{-1,0,0,\perp_2,\perp_2\})=\{-1,0,0,0\}$
- $mid_2(\{-1,-1,-1,0,1,\perp_1\})=-0.5$
- $center_3(\{-1,-1,0,1,\perp_1\})=-0.5$
- $center_2(\{-1,-1,0,1,\perp_1\})=-1/3$

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 [4]. *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 [4]. *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 $|V|=N$ such that at least $N-k$ entries of V are different from $\perp_r$ and lie in the interval $[a,b]$, and if $l\geq 2k$, then every entry of $chop_l^r(V)$ lies in $[a,b]$.*

Proof. Let $mult(\perp_r,V)=j$ (so $j\leq k$) and let Z denote the multiset of $2N-2j$ entries formed from $double(V)$ by removing all $2j$ copies of $\perp_r$. Now $chop_l^r(V)=red_{l-j}(Z)$, and at least $2N-2k$ entries of Z lie in $[a,b]$ so (since $l-j\geq 2k-2j$) 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 $mult(\perp_r,W)\leq k$. Let $|mult(v,V)-mult(v,W)|+|mult(w,V)-mult(w,W)|=m$. Then $|mult(v,chop_k^r(V))-mult(v,chop_k^r(W))|\leq m$ and $|mult(w,chop_k^r(V))-mult(w,chop_k^r(W))|\leq m$.*

Proof. Without loss of generality we may assume $v<w$.

If $k=0$, then every entry of each of V and W is one of v or w, so $(mult(v,V)-mult(v,W))=-(mult(w,V)-mult(w,W))$. In this case however $chop_k^r$ is just the operator *double*, so $|mult(v,chop_k^r(V))-mult(v,chop_k^r(W))|=2|mult(v,V)-mult(v,W)|=|mult(v,V)-mult(v,W)|+|mult(w,V)-mult(w,W)|=m$, and similarly $|mult(w,chop_k^r(V))-mult(w,chop_k^r(W))|=m$.

So suppose $k\geq 1$. We use induction on m. If $m=0$ then $V=W$ and there is nothing to prove. If $m=1$ then $|mult(\perp_r,V)-mult(\perp_r,W)|=1$, and we can assume without loss of generality that $mult(\perp_r,V)-mult(\perp_r,W)=1$, so that V is formed from W by by removing a single entry with value z (which is either v or w) and replacing it with an entry with value $\perp_r$. Let us put $j=mult(\perp_r,W)$ (so $j\leq k-1$) and let Z denote the multiset of $2N-2j$ entries formed by removing all occurrences of $\perp_r$ from $double(W)$. Now $chop_k^r(W)$ is formed from Z by removing the $k-j$ highest entries and the $k-j$ lowest entries. On the other hand, $chop_k^r(V)$ is formed from Z by removing two entries with value 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 $chop_k^r(V)$ from $chop_k^r(W)$ either by removing the $(k-j+1)$-st lowest entry of Z and replacing it with the $(k-j)$-th highest entry of Z, or else by replacing the $(k-j+1)$-st highest entry of Z by the $(k-j)$-th lowest entry of Z. In either case we see that the multiplicities of v in $chop_k^r(V)$ and $chop_k^r(W)$ can differ by at most 1 (and similarly for the multiplicities of w).

So suppose $m>1$. Without loss of generality we may assume $mult(\perp_r,W)\geq mult(\perp_r,V)$. We will construct a multiset W' so that $|W'|=N$, every entry of W' is one of v,w or $\perp_r$, $mult(\perp_r,W')\leq k$, $|mult(v,V)-mult(v,W')|+|mult(w,V)-mult(w,W')|\}=m-1$ and $|mult(v,W')-mult(v,W)|+|mult(w,W')$

$-mult(w, W)|=1$. The construction of W' will be different in two cases, depending on $mult(\perp_r, W)$. If $mult(\perp_r, W)>0$ then there is a value z (either v or w) so that $mult(z, W)<mult(z, V)$. In this case we let W' be formed from W by removing one entry with value $\perp_r$ and replacing it with z. If $mult(\perp_r, W)=mult(\perp_r, V)=0$ then there is a value z (either v or w) so that $mult(z, W)>mult(z, V)$. In this case we let W' be formed from W by removing one entry with value z and replacing it with $\perp_r$ (this does not violate the requirement on the multiplicity of $\perp_r$ in W' since $k\geq 1$). In either case, we can apply the induction hypothesis to the multisets V and W', and also to the multisets W' and W. We deduce that $|mult(v, chop_k^r(V))-mult(v, chop_k^r(W'))|\leq m-1$ and $|mult(v, chop_k^r(W'))-mult(v, chop_k^r(W))|\leq 1$. Therefore $|mult(v, chop_k^r(V))-mult(v, chop_k^r(W))|<m$, as required. Similarly $|mult(w, chop_k^r(V))-mult(w, chop_k^r(W))|\leq m$. 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 for each of the remaining m entries, the values 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 $r>1$, and all real entries of $V\cup W$ lie in the interval $[a, b]$. Let $m=\sum_{v\neq\perp_1}|mult(v, V)-mult(v, W)|$. 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 $[a, b]$.

If $k=0$, then every entry of each of V and W is real. Therefore if we let X denote the set of real numbers v such that $mult(v, V)>mult(v, W)$ and Y denote the set of real numbers v such that $mult(v, V)<mult(v, W)$, then we see that $\sum_{v\in X}(mult(v, V)-mult(v, V)-mult(v, W))=\sum_{v\in Y}(mult(v, W)-mult(v, V))$

$$=\frac{1}{2}\sum_v|mult(v, V)-mult(v, W)| \quad \text{since} \quad |V|=|W|.$$

Now in this case $center_k$ is just the operator $mean\circ double=mean$. Thus

$$center_0(V)-center_0(W)$$

$$=\frac{1}{N}\left(\sum_v v\cdot mult(v, V)-\sum_v v\cdot mult(v, W)\right)$$

$$=\frac{1}{N}\sum_v v(mult(v, V)-mult(v, W))$$

$$=\frac{1}{N}\left(\left(\sum_{v\in X} v(mult(v, V)-mult(v, W))\right)-\sum_{v\in Y} v(mult(v, W)-mult(v, V))\right)$$

$$\leq\frac{1}{N}\left(\left(\sum_{v\in X} b(mult(v, V)-mult(v, W))\right)-\left(\sum_{v\in Y} a(mult(v, W)-mult(v, V))\right)\right)$$

$$=\frac{1}{N}(b-a)\frac{1}{2}\sum_v|mult(v, V)-mult(v, W)|$$

$$=\frac{(b-a)\,m}{2N}$$

and by symmetry

$$center_0(W)-center_0(V)\leq(b-a)\,m/2N,$$

so

$$|center_0(V)-center_0(W)|\leq(b-a)\,m/2N,$$

as required.

So suppose $k\geq 1$. We use induction on m. If $m=0$ then $V=W$ and there is nothing to prove. If $m=1$ then $|mult(\perp_1, V)-mult(\perp_1, W)|=1$, and we can assume without loss of generality that $mult(\perp_1, V)-mult(\perp_1, W)=1$, so that V is formed from W by removing a single entry with value z and replacing it with an entry with value $\perp_1$. Let us put $j=mult(\perp_1, W)$ (so $j\leq k-1$) and let Z denote the multiset of $2N-2j$ entries formed by removing all occurrences of $\perp_1$ from $double(W)$. Let a' denote the value of the $(k-j)$-th lowest entry

of Z and let b' denote the value of the $(k-j)$-th highest entry of Z. Now $chop_k^1(W)$ is formed from Z by removing the $k-j$ highest entries and the $k-j$ lowest entries. On the other hand, $chop_k^1(V)$ 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 $z \leq a'$, this is equivalent to removing the $k-j-1$ highest and $k-j+1$ lowest entries from Z, while if $z \geq b'$ 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 $chop_k^1(V)$ from $chop_k^1(W)$ either by removing the $(k-j+1)$-th lowest entry of Z and adding b', or else by replacing the $k-j+1$ highest entry of Z by a'. Clearly in these cases, the sum of the entries of $chop_k^1(v)$ differs from the sum of the entries of $chop_k^1(W)$ by the difference of two elements of the interval $[a, b]$ which has absolute value at most $b-a$. In the remaining case we have $a' < z < b'$, but $chop_k^1(V)$ is obtained from $chop_k^1(W)$ by removing two entries with value z and replacing them with a' and b', which will alter the sum of the entries by $b'+a'-2z$. However $b'+a'-2z \leq b'+a'-2a' = (b'-a') \leq (b-a)$, and also $b'+a'-2z \geq b'+a'-2b' = -(b'-a') \geq -(b-a)$, so $|b'+a'-2z| \leq b-a$. Thus in every case

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

as required.

So suppose $m > 1$. Without loss of generality we may assume $mult(\perp_1, W) \geq mult(\perp_1, V)$. We will construct a multiset W' so that $|W'| = N$, $mult(\perp_1, W') \leq k$, $mult(\perp_r, W') = 0$ for $r > 1$, all real entries of W' lie in the interval $[a, b]$, $\sum_{v \neq \perp_1} |mult(v, V) - mult(v, W')| = m-1$ and $\sum_{v \neq \perp_1} |mult(v, W') - mult(v, W)| = 1$. The construction of W' will be different in two cases, depending on $mult(\perp_1, W)$. If $mult(\perp_1, W) > 0$ then there is a value z (in $[a, b]$) so that $mult(z, W) < mult(z, V)$. In this case we let W' be formed from W by removing one entry with value $\perp_1$ and replacing it with z. If $mult(\perp_r, W) = mult(\perp_r, V) = 0$ then there is a value z (in $[a, b]$) so that $mult(z, W) > mult(z, V)$. In this case we let W' be formed from W by removing one entry with value z and replacing it with $\perp_1$ (this does not violate the requirement on the multiplicity of $\perp_1$ in W' since $k \geq 1$). In either case, we can apply the induction hypothesis to the multisets V and W', and also to the multisets W' and W. We deduce that

$$|center_k(V) - center_k(W')| \leq (m-1)(b-a)/(2N-2k)$$

and

$$|center_k(W') - center_k(W)| \leq (b-a)/(2N-2k).$$

Therefore

$$|center_k(V) - center_k(W)| \leq |center_k(V) - center_k(W')| + |center_k(W') - center_k(W')| \leq m(b-a)/2N-2k,$$

as required. Q.E.D.

3 The crash-failure model: the algorithm

Throughout this paper, for ease of exposition 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 crash-failure model, our algorithm will require only $n > t$. The one round algorithm is given first. In this all processors exchange their initial values. Each processor forms a collection of initial values (representing those it has not heard about by $\perp_1$) and then applies the operator $center_t$ to this collection to produce its new final value. In detail, processor p, until it fails, must perform the following –

- Broadcast $v(p)$, and denote by $v(q_1, p)$ the value received by p from q_1. If the message from q_1 is missing set $v(q_1, p)$ to be $\perp_1$.
- Now p assigns $W(p)$ to be the multiset $\{v(1, p), v(2, p), \ldots, v(n, p)\}$.
- Finally processor p (unless it has previously crashed) decides on the final value $w(p) = center_t(W(p))$.

In an execution of this algorithm in the crash-failure model, let $Fail(1)$ denote the set of processors that crashed before sending any message, and let $Fail(2)$ denote the set that crashed before deciding on their final value. Let $l_1 = |Fail(2) \setminus Fail(1)| = |Fail(2)| - |Fail(1)|$. Also let U^1 denote the multiset $\{v(p): p \notin Fail(1)\}$ of initial values of processors that are active at the start of the communication, and let $U^2 = \{w(p): p \notin Fail(2)\}$, the final values of processors that reach a decision. We prove the following lemma that relates the final values to the initial values.

Lemma 7. *Suppose $n > t$. In an execution of the one-round algorithm above in the crash-failure model, such that $|Fault| \leq t$, we have*

$$\rho(U^2) \subseteq \rho(U^1)$$

and

$$\delta(U^2) \leq \frac{l_1}{2n-2t}\,\delta(U^1).$$

Proof. Let p_0 and p_1 be two processors that reach decisions. If q is a processor in $Fail(1)$, then $v(q, p_0) = v(q, p_1) = \perp_1$, since q crashed before sending any messages. If $q \notin Fail(2)$ then $v(q, p_0) = v(q, p_1) = v(q)$, since q sent correctly to all processors. (Since at least $n-t$ processors are non-faulty and have real initial values, we see from this that the multiplicity of $\perp_1$ in each of $W(p_0)$ and $W(p_1)$ is at most t). Finally, if $q \in (Fail(2) \setminus Fail(1))$, then either the two values $v(q, p_0)$ and $v(q, p_1)$ are equal or else one out of the two values is $\perp_1$. Thus the quantity $\sum_{v \neq \perp_1} |mult(v, W(p_0)) - mult(v, W(p_1))|$ is at most $|Fail(2) \setminus Fail(1)| = l_1$. If we apply Lemma 6, with $V = W(p_0)$, $W = W(p_1)$, $N = n$, $k = t$, and $[a, b] = \rho(U^1)$, we obtain the claimed results. Q.E.D.

Thus the algorithm above satisfies the validity condition, and since $l_1 \leq t$ we see that it has performance $K \leq \frac{t}{2n-2t}$. Asymptotically this is twice as good as the performance of the synchronous algorithm of [4]. (The intuitive reason behind this improved performance is that one Byzantine fault or two crashes produce similar differences between processors' views.)

When S rounds of communication are available, we can iterate the algorithm given above, by using the final values from each round (except the last) as initial values in the next round, and then deciding on the final values from the last round. Thus we have the following algorithm, where we give different names to the variables in different rounds to make the analysis clearer. Processor p, until it fails, must perform the following –

- Set $v^1(p) = v(p)$.
- For $r = 1, \ldots, S$ successively:
  - Broadcast $v^r(p)$, and denote by $v^r(q_1, p)$ the value received by p from q_1 in this round. If the message from q_1 is missing set $v^r(q_1, p)$ to be $\perp_1$.
  - Now p assigns $W^r(p)$ to be the multiset $\{v^r(1, p), v^r(2, p), \ldots, v^r(n, p)\}$.
  - Then processor p assigns $v^{r+1}(p) = center_t(W^r(p))$.
- Finally processor p (unless it has previously crashed) decides on the final value $w(p) = v^{S+1}(p)$.

Theorem 8. *Suppose $n > t$. Then the iterated algorithm given above is valid in the crash-failure model with performance*

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

Proof. Consider any execution of the iterated algorithm, in which $|Fault| \leq t < n$. To analyze this execution, we will denote by $Fail(r)$ the set of processors that crashed before sending any message in round r, for $r = 1, \ldots, S$, and by $Fail(S+1)$ the set of processors that crashed before reaching a decision. Notice that $Fail(r) \subseteq Fail(r+1)$. We let $l_r = |Fail(r+1) \setminus Fail(r)| = |Fail(r+1)| - |Fail(r)|$. Thus l_r is the number of processors that crashed during the r-th round of the algorithm. We let U^r denote the multiset $\{v^r(q) : q \notin Fail(r)\}$. Since each round is merely an application of the one-round algorithm, the previous lemma shows that $\rho(U^{r+1}) \subseteq \rho(U^r)$ and $\delta(U^{r+1}) \leq \frac{l_r}{2n-2t}\,\delta(U^r)$ for each r. Thus we deduce $\rho(U^{S+1}) \subseteq \rho(U^1)$ and $\delta(U^{S+1}) \leq \frac{l_1 \cdot l_2 \ldots l_S}{(2n-2t)^S}\,\delta(U^1)$. However $l_1 + l_2 + \ldots + l_S = |Fail(S+1)| - |Fail(1)|$, which is at most t. Since U^1 is a subset of the multiset of initial values, and the multiset of final values of correct processors is a subset of U^{S+1}, this proves that the iterated algorithm is valid with the claimed performance. Q.E.D.

In § 8 we prove a lower bound on any S-round algorithm for solving approximate agreement in the crash-failure model. This will show that as $n/t \longrightarrow \infty$ the iterated algorithm given in this section is asymptotically optimal.

4 The failure-by-omission and byzantine failure models: The algorithms

We now describe, for both remaining failure models, an algorithm that provides t-resilient approximate agreement for n processors using S rounds of communication (the values of t, n and S are "hard-wired" into the algorithms). The algorithms given are variants on a single plan. Thus we first describe this plan, using unspecified functions $\mathbf{Det}_r$, $\mathbf{F}_r$ and $\mathbf{G}$ for detecting faults and computing results. The specific algorithm for each failure model is then given by explaining which function is to be used for each of these. Recall that for ease of exposition we suppose that when a processor broadcasts information it sends to itself as well as to the other processors.

The algorithm has two phases. First there are S rounds of communication. During the r-th round of communication, the algorithm requires processor p to broadcast information it holds in the array $\tilde{v}(p_1, \ldots, p_{r-1}, p)$ and to collect the information sent to it in an array $v(p_1, \ldots, p_r, p)$. It then tries to deduce which processors are faulty, and next 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 method processor p uses to detect that process q is faulty is to examine (with a predicate $\mathbf{Det}_r$) the n values which reach p representing some information that was broadcast by q and then relayed to p by each recipient.

After the S rounds of communication, a processor will have an array of n^S values to operate on. The second phase now begins. The values are considered as forming n^S multisets, where $W(q_1, \ldots, q_S, p)$ contains the single value $v(q_1, \ldots, q_S, p)$. In successive steps, several multisets are combined into one, with elimination of extreme values. Thus in the $(S-r)$-th combination step processor p builds for each choice of $q_1, \ldots, q_r$ a multiset $W(q_1, \ldots, q_r, *, \ldots, *, p)$ by applying a function $\mathbf{F}_r$ to the union of the n multisets $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$, which were constructed at the previous step. At last a single large multiset $W(p)$ is obtained as the union of the multisets $W(q_1, *, \ldots, *, p)$, and then this is "averaged" using a function $\mathbf{G}$ to give the processor's new value $w(p)$.

Formally, the algorithm requires processor p to 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 message 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 $\langle \tilde{v}(q_1, q_2, \ldots, q_{r-1}, p)$: 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 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 of n values $\{v(q_1, \ldots, q_{r-1}, 1, p), v(q_1, \ldots, q_{r-1}, 2, p), \ldots, v(q_1, \ldots, q_{r-1}, n, p)\}$. If this multiset satisfies the predicate $\mathbf{Det}_r$ 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 $Fault(p, r) = Fault(p, r-1) \cup \{q: q \text{ has been detected as faulty by } p \text{ in round } r\}$.
  - Set $\tilde{v}(q_1, \ldots, q_{r-1}, q_r, p)$
$$= \begin{cases} v(q_1, \ldots, q_{r-1}, q_r, p) & \text{if } q_r \notin Fault(p, r) \\ \perp_r & \text{if } q_r \in Fault(p, r) \end{cases}$$
- 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
$$W(q_1, \ldots, q_r, *, \ldots, *, p)$$
$$= \mathbf{F}_r\left(\bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)\right)$$

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 $W(p) = \bigcup_{q_1=1}^{n} W(q_1, *, \ldots, *, p)$.
- Finally processor p should decide on its final value $w(p) = \mathbf{G}(W(p))$.

In the Byzantine failure model, our algorithm will require $n > 4t$. We say that a multiset of n values satisfies $\mathbf{Det}_r$ if no value has multiplicity at least $n-t$. We let $\mathbf{F}_r = red_{(n-4t)^{S-r-1}2t}$ and $\mathbf{G} = mid_{(n-4t)^{S-1}t}$.

In the failure-by-omission model, our algorithm will require $n > 2t$. We say that a multiset of n values satisfies $\mathbf{Det}_r$ if some entry is $\perp_{r-1}$. We let $\mathbf{F}_r = chop^{r+1}_{(2n-4t)^{S-r-1}2t}$ and $\mathbf{G} = center_{(2n-4t)^{S-1}t}$.

The rigorous analysis of the algorithms will be given in the following sections of the paper. Here, we try to indicate the essential reasons why the algorithms work well. The value $\tilde{v}(q_1, \ldots, q_S, p)$ ought to be the initial value of processor q_1 as passed from q_1 to q_2 in round 1, then from q_2 to q_3 in round 2, and so on, till q_S passed it to p in round S. Thus it ought to be equal to the value $\tilde{v}(q_1, \ldots, q_r)$ that processor q_r ought to have sent in round r. It may not be that value only if one of the processors q_i with $r \le i \le S$ (along the

way from q_r to p) omitted to send the value, or (in the Byzantine failure model) lied about it, or if the next processor q_{i+1} detected q_i as faulty and therefore replaced the value received by $\perp_i$. Thus $\tilde{v}(q_1, \ldots, q_r, \ldots, q_S, p)$ is $\tilde{v}(q_1, \ldots, q_r)$ if every processor q_i for $r \leq i \leq S$ is correct. Now the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p)$ is formed (by repeatedly combining, duplicating or removing extreme values) out of the collection of values $\langle \tilde{v}(q_1, \ldots, q_S, p)$: $q_i \in \{1, \ldots, n\}$ for all $i > r\rangle$. Thus $W(q_1, \ldots, q_r, *, \ldots, *, p)$ ought to contain many copies of the value $\tilde{v}(q_1, \ldots, q_r)$. In each of our algorithms we choose $\mathbf{F}_r$ to remove enough extreme values to ensure that, unless q_r behaved very badly before or during round r, $W(q_1, \ldots, q_r, *, \ldots, *, p)$ will contain no entries except copies of the value $\tilde{v}(q_1, \ldots, q_r)$. As a consequence the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p_0)$, computed by p_0, and the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p_1)$, computed by p_1, are identical unless q_r behaved very badly before or during round r. On the other hand, if q_r behaved very badly before round r, we ensure that the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p)$ will contain no entries except copies of $\perp_r$, and so again the multisets computed by different processors will be identical. To do this, we ensure that either q_r has crashed before the start of round r (so the intended recipients will receive nothing and set the corresponding values to be $\perp_r$) or else many processors have detected q_r as faulty before or during round r (so that any value sent by q_r during round r will be replaced by $\perp_r$ by the recipient). Thus the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p_0)$ computed by p_0, and the multiset $W(q_1, \ldots, q_r, *, \ldots, *, p_1)$ computed by p_1 are identical except when q_r behaved very badly for the first time during round r. Furthermore, we ensure that even in this case the two computed multisets have many entries in common. This in turn ensures that the multisets $W(p_0)$ and $W(p_1)$ have many entries in common, and that therefore the chosen final values $w(p_0)$ and $w(p_1)$ are close together.

5 The failure-by-omission model: analysis

When discussing the algorithm for the failure-by-omission model, we will need to assume that $n > 2t$, since otherwise the algorithm given is meaningless, involving as it does operators such as $\mathbf{G} = center_{(2n-4t)^{S-1}t}$.

For any execution of the algorithm in the failure-by-omission model, for each $r = 1, \ldots, S$ we let $Fail(r)$ denote the set of processors that have crashed before sending any of the messages in round r. Let

$$Exposed(r) = Fail(r) \cup \bigcap_{p \notin Fail(r+1)} Fault(p, r).$$

Also as a convention we set

$$Exposed(S+1) = \{1, \ldots, n\} \setminus Corr.$$

Note that $Exposed(r) \subseteq Exposed\ (r+1)$. We put $l_r = |Exposed(r+1) \setminus Exposed(r)| = |Exposed(r+1)| - |Exposed(r)|$. We will see that l_r is the number of processors whose messages in round r cause differences between other processors' views.

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 show that the fault detection procedure never makes a mistake.

Lemma 9. *Suppose $n > 2t$. In an execution of the algorithm in the failure-by-omission model, for which $f \leq t$, if $p \notin Fail(r+1)$ and $q \in Corr$, then $q \notin Fault(p, r)$.*

Proof. We use 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 $v(q_1, \ldots, q_{r-2}, q, q_r, p) = \tilde{v}(q_1, \ldots, q_{r-2}, q, q_r)$. But $\tilde{v}(q_1, \ldots, q_{r-2}, q, q_r) = v(q_1, \ldots, q_{r-2}, q, q_r)$ 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 $q \notin Fault(p, r)$. Q.E.D.

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 $Exposed(r+1) \setminus Exposed(r)$ can cause different processors to choose different representatives for a round r value.

Lemma 10. *Suppose $n > 2t$. In any execution of the algorithm in the failure-by-omission model, 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*

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

Proof. 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 $q_S \in Fault(p, S)$ then $\tilde{v}(q_1, \ldots, q_S, p)=\perp_S$, while otherwise $\tilde{v}(q_1, \ldots, q_S, p)=v(q_1, \ldots, q_S, p)$ which is either equal to $\perp_S$ (if q_S did not send its round S message to p) or to $\tilde{v}(q_1, \ldots, q_S)$, since in the failure-by-omission model any value that is sent is correct.

(ii): If $q_S \notin Exposed(S+1)$ then $q_S \in Corr$. We deduce that q_S did send its round S message to p and also that $q_S \notin Fault(p, S)$. 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 $q_S \in Exposed(S)$ then either $q_S \in Fail(S)$ (so q_S did not send its round S message to p), or else $q_S \in Fault(p, S)$. 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 $l_{r+1} \ldots l_S$ 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

$|mult(v(q_1, \ldots, q_r), W(q_1, \ldots, q_r, *, \ldots, *, p_0))$
$-mult(v(q_1, \ldots, q_r), W(q_1, \ldots, q_r, *, \ldots, *, p_1))|$
$\leq l_{r+1} \cdot l_{r+2} \ldots l_S$

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 $q_r \in Fault(q_{r+1}, r)$) 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 $q_{r+1} \in Exposed(r+1)$, 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 $\bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ 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 $Exposed(r+2)$ (by (ii) for $r+1$). Thus when we apply $chop_k^{r+1}$, where $k=2t(2n-4t)^{S-r-1}$, to

$$\bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_{r+1}, *, \ldots, *, p)$$

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 $q_r \notin Exposed(r+1)$ then for every $q \in Corr$, $\tilde{v}(q_1, \ldots, q_r, q)=\tilde{v}(q_1, \ldots, q_r)$. This is proved by contradiction: suppose there is $q_{r+1} \in Corr$ with $\tilde{v}(q_1, \ldots, q_r, q_{r+1})=\perp_r$ and therefore for any $p \notin Fail(r+2)$ 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 $p \notin Fail(r+2)$ contradicts the assumption $q_r \notin Exposed(r+1)$. Now by (ii) for $r+1$, if $q_r \notin Exposed(r+1)$, $q_{r+1} \in Corr$ and $p \notin Crash$, 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 $q_r \notin Exposed(r+1)$ and $p \notin Crash$, then $\bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ 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 $q_r \in Exposed(r)$ then for every $q \in Corr$, $\tilde{v}(q_1, \ldots, q_r, q)=\perp_r$ (if $q_r \in Fault(q, r)$ this is explicit in the algorithm, and if $q_r \in Fail(r)$ 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 $q_r \in Exposed(r)$, $q_{r+1} \in Corr$ and

$p \notin Crash$, 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 $q_r \in Exposed(r)$ and $p \notin Crash$, then $\bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ 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 $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_0) = W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_1)$ if $q_{r+1} \notin Exposed(r+2) \setminus Exposed(r+1)$, 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

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

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 (using the triangle inequality) that

$$\left| mult\left(\tilde{v}(q_1, \ldots, q_r), \bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_0)\right) - mult\left(\tilde{v}(q_1, \ldots, q_r), \bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_1)\right)\right| + \left| mult\left(\perp_r, \bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_0)\right) - mult\left(\perp_r, \bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_1)\right)\right| \leq l_{r+1} \cdot l_{r+2} \ldots l_S.$$

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

Theorem 11. *If $n > 2t$, then the algorithm in the failure-by-omission model is valid and has performance*

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

Proof. We have by (ii) and (iii) of Lemma 10 for $r=1$ that $W(q_1, *, \ldots, *, p_0) = W(q_1, *, \ldots, *, p_1)$ unless $q_1 \in Exposed(2) \setminus Exposed(1)$. For these l_1 values of q_1 we have by (iv) for $r=1$ that

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

since $\tilde{v}(q_1) = v(q_1)$. We can apply Lemma 6 with

$$V = \bigcup_{q_1=1}^{n} W(q_1, *, \ldots, *, p_0),$$

$$W = \bigcup_{q_1=1}^{n} W(q_1, *, \ldots, *, p_1),$$

$N = n(2n-4t)^{S-1}$, $k = t(2n-4t)^{S-1}$ 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

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

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 $l_1 + l_2 + \ldots + l_S = |Exposed(S+1)| - |Exposed(1)| \leq t$. This proves that our algorithm has, as claimed, performance

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

Q.E.D.

6 The byzantine failure model: analysis

When discussing the algorithm for the byzantine failure model, we will need to assume that $n > 4t$, since otherwise the algorithm given is meaningless, involving as it does operators such as $\mathbf{G} = mid_{(n-4t)^{S-1}t}$.

In discussing an execution of the algorithm for the Byzantine failure model, we will extend the definition of $Fault(p, r)$ to include $r=0$ and $r=S+1$, by setting $Fault(p, 0) = \emptyset$, and $Fault(p, S+1) = \{1, \ldots, n\} \setminus Corr$ as a convention. We put $Exposed(r) = \bigcap_{p \in Corr} Fault(p, r)$ and $l_r = |Exposed(r+1) \setminus Exposed(r)| = |Exposed(r+1)| - |Exposed(r)|$. 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 whose messages in round r that will cause differences between other processors' views.

First we observe that the fault detection procedure never makes a mistake.

Lemma 12. *Suppose $n > 4t$. In an execution of the algorithm for the Byzantine failure model, for which $f \leq t$, if $p \in Corr$ and $q \in Corr$, then $q \notin Fault(p, r)$.*

Proof. This is immediate for the special cases $r=0$ and $r=S+1$ defined above. For the other values of r, we use proof by induction on r. If $r=1$, and $p\in Corr$, $q\in Corr$ then $q\notin Fault(p,1)$ as $Fault(p,1)=\emptyset$.

Now for arbitrary r suppose $p\in Corr$ and $q\in Corr$. If $q_r\in Corr$ then by the induction hypothesis $q\notin Fault(q_r, r-1)$ 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 has value $\tilde{v}(q_1, \ldots, q_{r-2}, q)$, and so the multiset does not satisfy the predicate $\mathbf{Det}_r$.

Therefore q is not detected as faulty by p in round r, but by the induction hypothesis $q\notin Fault(p, r-1)$. Thus we see $q\notin Fault(p,r)$ as required. Q.E.D.

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 $Exposed(r+1)\setminus Exposed(r)$ will cause differences between the multisets computed by different correct processors to represent the same round r value.

Lemma 13. *Suppose $n>4t$. In an execution of the algorithm for the Byzantine failure model, for which $f\leq t$, we can conclude:*

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

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

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

(iv): *If $p_0\in Corr$ and $p_1\in Corr$, then $|W(q_1, \ldots, q_r, *, \ldots, *, p_0) \cap W(q_1, \ldots, q_r, *, \ldots, *, p_1)| \geq (n-4t)^{S-r} - l_{r+1}\cdot l_{r+2}\ldots l_S$.*

Proof. We use descending induction on r. First, suppose $r=S$.

(i): If $q_S\in Corr$ and $p\in Corr$, then $W(q_1, \ldots, q_S, p) = \{\tilde{v}(q_1, \ldots, q_S, p)\}$, but $q_S\notin Fault(p, S)$ 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 $q_S\notin Exposed(S+1)$, then by definition of the sets $Fault(q, S+1)$, we must have $q_S\in Corr$ and so, by (i) proved above, if $p_0\in Corr$ and $p_1\in Corr$, 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 $q_S\in Exposed(S)$ and $p_0\in Corr$, then $q_S\in Fault(p_0, S)$ 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 $(n-4t)^{S-r} - l_{r+1}\ldots l_S$ 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 $|W(q_1, \ldots, q_S, p_0) \cap W(q_1, \ldots, q_S, p_1)| \geq (n-4t)^{S-r} - l_{r+1}\ldots l_S$ in this case.

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

(i): If $q_r\in Corr$ and $p\in Corr$, then for $q_{r+1}\in Corr$, 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 $q_r\in Corr$ (and so $q_r\notin Fault(q_{r+1}, r)$), $\tilde{v}(q_1, \ldots, q_r, q_{r+1}) = v(q_1, \ldots, q_r, q_{r+1}) = \tilde{v}(q_1, \ldots, q_r)$. Thus the combined multiset $\bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p)$ contains at least $(n-4t)^{S-r-1}(n-t)$ entries each of which has value $\tilde{v}(q_1, \ldots, q_r)$, 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 $q_r\notin Exposed(r+1)$ then the multiset $\{\tilde{v}(q_1, \ldots, q_r, q): q\in Corr\}$ has some value[2] (say v) occurring with multiplicity at least $n-2t$. This is proved by contradiction: suppose that there is a choice of $q_1, \ldots, q_r$ so that the multiset $\{\tilde{v}(q_1, \ldots, q_r, q): q\in Corr\}$ has every value with multiplicity less than $n-2t$. For $p\in Corr$ and $q\in Corr$, $v(q_1, \ldots, q_r, q, p) = \tilde{v}(q_1, \ldots, q_r, q)$ so the multiset $\{v(q_1, \ldots, q_r, 1, p), v(q_1, \ldots, q_r, 2, p), \ldots, v(q_1, \ldots, q_r, n, p)\}$ has every value with multiplicity less

[2] Since $n>4t$, there is at most one value with multiplicity at least $n-2t$

than $n-t$, and so $q_r \in Fault(p, r+1)$, but this holds for all correct p which would contradict $q_r \notin Exposed(r+1)$.

Now if $q_{r+1} \in Corr$ and $p_0 \in Corr$, by (i) for $r+1$ as above, $W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_0)$ consists of $(n-4t)^{S-r-1}$ copies of $\tilde{v}(q_1, \ldots, q_r, q_{r+1})$. Thus in this situation $\bigcup_{q_{r+1}=1}^{n} W(q_1, \ldots, q_r, q_{r+1}, *, \ldots, *, p_0)$ contains at least $(n-4t)^{S-r-1}(n-2t)$ entries each of which has value v (namely $(n-4t)^{S-r-1}$ for each of at least $(n-2t)$ different choices of q_{r+1} satisfying $\tilde{v}(q_1, \ldots, q_r, q_{r+1}) = v$). By Lemma 1, all $(n-4t)^{S-r}$ entries of $W(q_1, \ldots, q_r, *, \ldots, *, p_0)$ have value v. If $p_1 \in Corr$ then similarly $W(q_1, \ldots, q_r, *, \ldots, *, p_1)$ consists of $(n-4t)^{S-r}$ copies of v. So these multisets are equal.

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

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

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

Thus by Lemma 2

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

We can now prove the claimed upper bound on performance of our algorithm for the Byzantine failure model.

Theorem 14. *If $n > 4t$, then the algorithm for the Byzantine failure model is valid and has performance*

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

Proof. When we apply Lemma 13 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 have value $v(q_1)$.

(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

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

Now if $p \in Corr$ we see that $\bigcup_{q_1=1}^{n} W(q_1, *, \ldots, *, p)$ contains at least $(n-t)(n-4t)^{S-1}$ entries in the range $\rho(\hat{U})$ 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

$$|W(p_0) \cap W(p_1)| \geq (n-l_1)(n-4t)^{S-1} + l_1((n-4t)^{S-1} - l_2 \ldots l_S) = n(n-4t)^{S-1} - l_1 l_2 \ldots l_S$$

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

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

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 $l_1 + l_2 + \ldots + l_S = |Exposed(S+1)| = |Fault| \leq t$. This proves that our algorithm has, as claimed, performance

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

Q.E.D.

It is interesting to note that for $S=2$ our algorithm therefore gives an implementation of Crusader's Agreement [3] 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[3]. In fact our implementation has a stronger property: if any p_0 fails to detect that q is faulty, then all those p that do detect it know what value p_0 has chosen.

7 The Byzantine failure model: A lower bound

This section gives a detailed account of a lower bound on achievable performance for any S-round, t-resilient approximate agreement algorithm in the Byzantine failure model[4].

Theorem 15. *Suppose $n>t+1$. Any algorithm that performs valid t-resilient approximate agreement in the Byzantine failure model using at most S rounds of communication, has performance*

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

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 $\langle u(q_1, \ldots, q_{r-1}, p)$: 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 $w(p)$.

[3] A similar use of Inexact Agreement to implement Crusader Agreement was given in [12]

[4] The asymptotic form of this lower bound was given in [4]

Different algorithms are given by different choices of the function f. Notice that any algorithm (like those given in this paper) 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 $l_1 + \ldots + l_S \leq t$. 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 $l_1 \times l_2 \times \ldots \times l_S$. 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 $f(M_{(1,\ldots,1)})=0$. Also we consider an execution where the processors $1, 2, \ldots, (m_1-1)l_1$ 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 $M_{(m_1,1,\ldots,1)}$ as their view, and the validity condition requires them to agree on 1 as their new value, so $f(M_{(m_1,1,\ldots,1)})=1$. Since the chain of arrays M_I reaches from $I=(1,\ldots,1)$ to $I=(m_1,1,\ldots,1)$ in $(m_1-1)m_2\ldots m_S$ steps, we get a chain of real numbers $f(M_I)$ reaching from 0 to 1 in $(m_1-1)m_2\ldots m_S$ steps. Thus there is some pair of consecutive values where

$$|f(M_I)-f(M_{I++})|\geq\frac{1}{(m_1-1)m_2\ldots m_S}$$
$$\geq\frac{1}{m_1 m_2\ldots m_S},$$
$$\text{so } K\geq\frac{1}{m_1\ldots m_S}.$$

Since $m_k=\lceil n/l_k\rceil\leq(n+l_k)/l_k\leq(n+t)/l_k$,

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

This is true for any choice of $l_1,\ldots,l_S$ with each l_i a nonnegative integer and $l_1+\ldots+l_S\leq t$. (Our argument above covers the cases when all the l_i are positive, but the inequality is trivially true if any l_i is zero.) Thus we have the claimed lower bound

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

All that remains is to fulfill our promise to give an execution with $M_{(i_1,i_2,\ldots,i_S)}$ as the view for some correct processor p_0, and $M_{(i_1,\ldots,i_S)++}$ as the view for a correct processor p_1. We give the construction, and leave the reader to verify that the processors have the views stated. 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.

We observe here that if $n=t+1$ there is the immediate lower bound that $K\geq 1$ for any t-resilient algorithm for approximate agreement in the Byzantine failure model. This follows from the fact that a correct processor may be the only correct processor, and so the validity condition requires that it must choose its new value equal to its initial value.

8 The 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 § 7) 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 § 7, but this time we will do so implicity by giving a recursive recipe for the execution that lies between successive views. This proof is very closely related to the proofs in [5] and [13] of the impossibility of exact agreement in fewer than $t+1$ rounds, and also to the proof in [2] 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 $m \geq k$.

We prove two preliminary lemmas that show that certain executions are k-similar.

Lemma 16. *Suppose $n > t+1$ and $S > 1$. Let $l_1, l_2, \ldots, l_S$ be any collection of positive integers such that $l_1 \geq l_2 \geq \ldots \geq l_S$ and $l_1 + \ldots + l_S \leq t$. Put $m_i = \lceil n/l_i \rceil$. 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 $N(r) = \prod_{j=r+1}^{S} 2m_j + 2$.*

Proof. We first remark that the statement is not necessarily true if $r = S$. Let J denote the set of indices of processors that fail in execution ρ during the first $r-1$ rounds and let J' denote the set of indices of processors that fail in execution ρ during round r. We denote by j the number of indices in j and by j' the number in J'. Thus $j < l_1 + \ldots + l_{r-1}$ and $j + j' < l_1 + \ldots + l_{r-1} + l_r$. We will use descending induction on r.

Suppose $r = S-1$. The discussion is divided into two cases, depending on whether $n > j + j' + 2l_S + 1$ or not.

If $n > j + j' + 2l_S + 1$, then for each $k = 1, \ldots, m_S$ let q_k be the greatest processor index that is not among $J \cup J'$ nor in the range $(k-2)\, l_S + 1, \ldots, k l_S$, and let p_k be the least processor index that is not among $J \cup J'$ nor in the range $(k-2)\, l_S + 1, \ldots, (k-1) l_S$. Now q_k is strictly greater than the least processor index that is not among $J \cup J'$ nor in the range $(k-2)\, l_S + 1, \ldots, k l_S$, (as this set contains at least two indices) and this in turn is greater than or equal to each of p_k and p_{k+1} each of which is the least index in a set containing all indices not in $J \cup J'$ nor in the range $(k-2)\, l_S + 1, \ldots, k l_S$. Thus $p_k < q_{k-1}$ and $p_k < q_k$. 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 with indices in J' 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 $(k-1) l_S + 1, \ldots, k l_S$ 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 with indices in J' do send to any processor with index $1, 2, \ldots, k l_S$ as well as those processors that they send to in ρ. In round S, each of the processors $(k-1) l_S + 1, \ldots, k l_S$ that has not failed earlier, fails after sending message 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} (the only difference between the executions lies in when processors fail in round S, but in each execution these failures occur after sending messages to p_k, which is thus unaware of the time of failure) so $\rho_{2(k-1)} \approx \rho_{2k-1}$. Also the view of q_k is the same in ρ_{2k-1} as in ρ_{2k} (since the only difference between the executions lies in which round $S-1$ messages reached processors $(k-1) l_S + 1, \ldots, k l_S$, and none of these sent a later message to q_k) 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 with index in J' as well as each of $(m_s - 1) l_S + 1, \ldots, n$ 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 ρ_{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, $\rho \sim^{2m_S+1} \hat{\rho}$, but $2m_S + 1 \leq N(S-1)$, so $\rho \sim^{N(S-1)} \hat{\rho}$.

If we have $n \leq j+j'+2l_S+1$ then the set of processors that are not in $J \cup J'$ has size $n-j-j' \leq 2l_S+1 \leq 3l_S$, but it also includes at least 3 processors, since $n > t+1 \geq l_1+\ldots+l_S+1 \geq j+j'+l_S+1 \geq j+j'+2$. Thus we can find indices p_1, p_2 and p_3 satisfying the following conditions: none of the indices p_1, p_2 or p_3 is in $J \cup J'$, the number of processor indices that are less than or equal to p_1 but not in $J \cup J'$ is at least one and at most l_S, the number of processor indices that are greater than p_1 and less than or equal to p_2 but not in $J \cup J'$ is at least one and at most l_S, the number of indices that are greater than p_2 and less than or equal to p_3 but not in $J \cup J'$ is at least one and at most l_S, and no processor index is greater than p_3 but not in $J \cup J'$. (Thus we divide the processors that do not fail in ρ into three groups lying in the intervals $1, \ldots, p_1$ and $p_1+1, \ldots, p_2$ and $p_2+1, \ldots, p_3$. Each group contains at most l_S processors.) Let ρ_1 be the execution which is identical to ρ during the first $S-1$ rounds and during round S each of the processors $1, \ldots, p_1$ (that has not failed earlier) fails after sending messages to processors $1, \ldots, p_2$. Let p_2 be the execution which is identical to ρ during the first $S-2$ rounds, and also during round $S-1$ except that the processors with indices in J' do send to each of the processors $1, \ldots, p_1$ as well as those they send to in ρ, and during round S each of the processors $1, \ldots, p_1$ (that has not failed earlier) fails after sending messages to processors $1, \ldots, p_2$. Let ρ_3 be the execution which is identical to ρ during the first $S-2$ rounds, and also during round $S-1$ except that the processors with indices in J' do send to each of the processors $1, \ldots, p_1$ as well as those they send to in ρ, and during round S no failures occur. Let ρ_4 be the execution which is identical to ρ during the first $S-2$ rounds, and also during round $S-1$ except that the processors with indices in J' do send to each of the processors $1, \ldots, p_1$ as well as those they send to in ρ, and during round S each of the processors $p_1+1, \ldots, p_2$ (that has not failed earlier) fails after sending messages to processors $1, \ldots, p_2$. Let ρ_5 be the execution which is identical to ρ during the first $S-2$ rounds, and also during round $S-1$ except that the processors with indices in J' do send to each of the processors $1, \ldots, p_2$ as well as those they send to in ρ, and during round S each of the processors $p_1+1, \ldots, p_2$ (that has not failed earlier) fails after sending messages to processors $1, \ldots, p_2$. Let ρ_6 be the execution which is identical to ρ during the first $S-2$ rounds, and also during round $S-1$ except that the processors with indices in J' do send to each of the processors $1, \ldots, p_2$ as well as those they send to in ρ, and during round S each of the processors $p_2+1, \ldots, p_3$ (that has not failed earlier) fails after sending messages to processors $1, \ldots, p_1$. Let ρ_7 be the execution which is identical to ρ during the first $S-2$ rounds, and also during round $S-1$ except that the processors with indices in J' do send to every processor (and so do not fail during round $S-1$) and during round S each of the processors $p_2+1, \ldots, p_3$ (that has not failed earlier) and also each processor with index in J' fails after sending messages to processors $1, \ldots, p_1$. The reader may check that the view of processor p_2 is the same in ρ_0 as in ρ_1, the view of processor p_3 is the same in ρ_1 as in ρ_2, the view of processor p_2 is the same in ρ_2 as in ρ_3, the view of processor p_1 is the same in ρ_3 as in ρ_4, the view of processor p_3 is the same in ρ_4 as in ρ_5, the view of processor p_1 is the same in ρ_5 as in ρ_6, the view of processor p_2 is the same in ρ_6 as in ρ_7, and the view of processor p_1 is the same in ρ_7 as in $\hat{\rho}$. Thus $\rho_0 \sim^8 \hat{\rho}$, but (since $l_1 \geq l_S$, $l_1+l_S \leq t$ and $n > t$) we have $n > 2l_S$ so $m_S \geq 3$ and $8 \leq 2m_S+2 = \mathrm{N(S-1)}$.

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 with indices in J' 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 by the end of each round 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 with indices in J' 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 with indices in J' 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

correct processor gets the same view in $p_{3(k-1)} \sim^{N(r+1)} p_{3k-2}$ so $p_{3k-1} \approx p_{3k-1}$. Further $p_{3m_{r+1}}$, in which processors with indices in J' 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 $\rho \sim^{(2m_{r+1}+1)N(r+1)+m_{r+1}} \hat{\rho}$. Since $l_{r+1} \geq l_{r+2}$ we deduce $m_{r+1} \leq m_{r+2} \leq N(r+1)$, so

$$(2m_{r+1}+1)N(r+1)+m_{r+1} \leq (2m_{r+1}+1)N(r+1)+N(r+1)=N(r).$$

Thus we see $\rho \sim^{N(r)} \hat{\rho}$. Q.E.D.

Lemma 17. *Suppose $n>t+1$. Let $l_1, l_2, \ldots, l_S$ be any collection of positive integers such that $l_1 \geq l_2 \geq \ldots \geq l_S$ and $l_1 + \ldots + l_S \leq t$. Put $m_i = \lceil n/l_i \rceil$. Let $\rho = \rho_0$ be the execution where all processors have initial value 0 and no failure occur, and let $\hat{\rho}$ be the execution where all initial values are 1 and no failures occur. Then $\rho \sim^N \hat{\rho}$ where $N \leq (2m_1+2)\cdot(2m_2+2)\ldots(2m_S+2)$.*

Proof. We will give separate proofs depending on the value of S and n.

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 16, $\rho_{3(k-1)} \sim^{N(1)} \rho_{3k-2}$ and $\rho_{3k-1} \sim^{N(1)} \rho_{3k}$. Also the view of every correct 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 $N = 2m_1 N(1) + m_1 \leq (2m_1+1)N(1)$ (since $m_1 \leq m_2 \leq N(1)$) and thus $N \leq \prod_{j=1}^{S} 2m_j + 2$.

Suppose $S=1$ and $n>2t+1$. For each $k=1, \ldots, m_1$ let q_k be the greatest processor index that is not in the range $(k-2)l_1, \ldots, kl_1$. Let p_k be the least processor index that is not in the range $(k-2)l_1+1, \ldots, (k-1)l_1$. We see that q_k is strictly greater than the least processor index that is not in the range $(k-2)l_1, \ldots, kl_1$ (since at least $n-2t \geq 2$ indices are not in the that range) and this in turn is greater than or equal to both p_k and p_{k+1}, each of which is the least index of a set containing all those not in the range $(k-2)l_1+1, \ldots, (k-1)l_1$. Thus we have $p_k < q_k$ 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} (as all processors have the same initial values in the two executions, and p_k receives a message from every processor in each execution) so $\rho_{2(k-1)} \approx \rho_{2k-1}$. Also the view of q_k is the same in ρ_{2k-1} as in ρ_{2k} (as q_k does not receive a message in either execution from those processors with initial values that are different in the two executions) 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 $N = 2m_1+1 \leq 2m_1+2$.

Suppose $S=1$ and $n=2t+1$. Let ρ_1 be the execution in which all processors have initial value 0 and during round 1 each of the processors $1, \ldots, t$ fails after sending messages to processors $1, \ldots, 2t$. Let ρ_2 be the execution in which processors $1, \ldots, t$ have initial value 1 and the others have initial value 0, and during round 1 each of the processors $1, \ldots, t$ fails after sending messages to processors $1, \ldots, 2t$. Let ρ_3 be the execution in which processors $1, \ldots, t$ have initial value 1 and the others have initial value 0, and no failures occur. Let ρ_4 be the execution in which processors $1, \ldots, t$ have initial value 1 and the others have initial value 0, and during round 1 each of the processors $t+1, \ldots, 2t$ fails after sending messages to processors $1, \ldots, 2t$. Let ρ_5 be the execution in which processors $1, \ldots, 2t$ have initial value 1 and processor n has initial value 0, and during round 1 each of the processors $t+1, \ldots, 2t$ fails after sending messages to processors $1, \ldots, 2t$. Let ρ_6 be the execution in which processors $1, \ldots, 2t$ have initial value 1 and processor n has initial value 0, and during round 1 processor n fails after sending messages to processors $1, \ldots, t$. Let ρ_7 be the execution in which all processors have initial value 1 and during round 1 processor n fails after sending messages to processors $1, \ldots, t$. The reader may check that the view of processor $t+1$ is the same in ρ_0 as in ρ_1, the view of processor n is the same in

ρ_1 as in ρ_2, the view of processor $t+1$ is the same in ρ_2 as in ρ_3, the view of processor 1 is the same in ρ_3 as in ρ_4, the view of processor n is the same in ρ_4 as in ρ_5, the view of processor 1 is the same in ρ_5 as in ρ_6, the view of processor $t+1$ is the same in ρ_6 as in ρ_7, and the view of processor 1 is the same in ρ_7 as in $\hat{\rho}$. Thus $\rho_0 \sim^8 \hat{\rho}$, but (since $l_1 \leq t$ and $n=2t+1$) we have $n \geq 2l_1+1$ so $m_1 \geq 3$, implying $8 \leq 2m_1+2$.

Finally, suppose $S=1$ and $t+2 \leq n \leq 2t$. Notice that this requires $t \geq 2$. Let ρ_1 be the execution in which all processors have initial value 0, and during round 1 processors $1, \ldots, t$ fail after sending messages to processors $1, \ldots, t+1$. Let ρ_2 be the execution in which processors $1, \ldots, t$ have initial value 1 while the others have initial value 0, and during round 1 processors $1, \ldots, t$ fail after sending messages to processors $1, \ldots, t+1$. Let ρ_3 be the execution in which processors $1, \ldots, t$ have initial value 1 while the others have initial value 0, and no failures occur. Let ρ_4 be the execution in which processors $1, \ldots, t$ have initial value 1 while the others have initial value 0, and during round 1 processors $t+1, \ldots, n$ fail after sending messages to processors $1, \ldots, t-1$. Let ρ_5 be the execution in which all processors have initial value 1, and during round 1 processors $t+1, \ldots, n$ fail after sending messages to processors $1, \ldots, t-1$. The reader may check that the view of processor $t+1$ is the same in ρ_0 as in ρ_1, the view of processor $t+2$ is the same in ρ_1 as in ρ_2, the view of processor $t+1$ is the same in ρ_2 as in ρ_3, the view of processor 1 is the same in ρ_3 as in ρ_4, the view of processor t is the same in ρ_4 as in ρ_5, and the view of processor 1 is the same in ρ_5 as in $\hat{\rho}$. Thus $\rho_0 \sim^6 \hat{\rho}$, but (since $l_1 \leq t$ and $n \geq t+2$) we have $n \geq l_1+2$ so $m_1 \geq 2$, implying $6 \leq 2m_1+2$. Q.E.D.

Now we can prove the lower bound for this failure model.

Theorem 18. *Suppose $n > t+1$. Any algorithm that performs valid t-resilient approximate agreement in the crash-failure model using at most S rounds of communication, has performance*

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

Proof. Let $l_1, l_2, \ldots, l_S$ be an arbitrary collection of nonnegative integers such that $l_1 + \ldots + l_S \leq t$. We will show that $K \geq \frac{l_1 \ldots l_S}{(2n+3t)^S}$. This clearly implies the theorem. If any l_i is zero, this inequality is trivially true. Otherwise we can rename the values so that $l_1 \geq l_2 \geq \ldots \geq l_S$, without affecting the sum or product of the values.

Let $\rho = \rho_0$ be the execution where all processors have initial value 0 and no failures occur, and let $\hat{\rho}$ be the execution where all initial values are 1 and no failures occur. Put $m_i = \lceil n/l_i \rceil$, and $N = (2m_1+2) \cdot (2m_2+2) \ldots (2m_S+2)$. Lemma 17 shows the existence of 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

$$N = \prod_{j=1}^{S} 2m_j + 2 \leq \prod_{j=1}^{S} (2n/l_j + 3)$$

we have

$$K \geq \prod_{j=1}^{S} l_j/(2n+3l_j) \geq \prod_{j=1}^{S} l_j/(2n+3t). \quad \text{Q.E.D.}$$

Since any algorithm for the failure-by-omission model works as well or better in the more restrictive crash-failure model, this lower bound also applies to a failure-by-omission system.

9 Conclusions

We have presented algorithms to solve S-round t-resilient approximate agreement in each of the crash-failure, failure-by-omission and Byzantine failure models. For fixed S and t, each algorithm has performance that is asymptotic as $n \longrightarrow \infty$ to the best possible in that model, by lower bounds proved in this paper.

These algorithms have the nice property that they can be easily modified so that they can provide tentative values that get closer and closer together, reaching exact agreement after $t+1$ rounds.

The results in this paper are summarized in the following Table:

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

For the crash-failure model, we iterate a one-round algorithm. The multi-round algorithms given for the failure-by-omission and Byzantine failure models involve processors exchanging information and then forming multisets $W(q_1, \ldots, q_r, *, \ldots, *, p)$ to represent all the information $\{v(q_1, \ldots, q_r, q_{r+1}, \ldots, q_S, p)$: $q_j = 1, 2, \ldots, n$ for $j > r\}$, using the operations of combining multisets and removing extreme values repeatedly to increase the amount of unanimity in each multiset. In order to achieve good performance each processor tries to detect which processors are faulty, and ignores messages sent by processors which have been detected.

Acknowledgments. I would like to thank Professor Nancy Lynch for teaching me about distributed algorithms and suggesting this problem, Michael Merritt for finding a major error in an early draft of this paper, Brian Coan and William Weihl for detailed comments on later drafts, Leslie Lamport for suggestions about the crash-failure case, Yoram Moses for fruitful discussions about the lower bounds, and two very thorough anonymous referees for many helpful suggestions.

References

1. Coan B (1986) Communication-efficient canonical form for fault-tolerant distributed protocols. Proc 5th ACM Symp on Principles of Distributed Computing, pp 63–72
2. Coan B, Dwork C (1986) Simultaneity is harder than agreement. Proc 5th Symp on Reliability in Distributed Software and Database Systems, pp 141–150
3. Dolev D (1982) The Byzantine generals strike again. J Algorithms 3:14–30
4. Dolev D, Lynch N, Pinter S, Stark E, Weihl W (1986) Reaching approximate agreement in the presence of faults. JACM 33(3):499–516
5. Dwork C, Moses Y (1986) Knowledge and common knowledge in a Byzantine environment I: crash failures. Proc 1986 Conf on Theoretical Aspects of Reasoning About Knowledge, pp 149–169
6. Fischer M (1983) The consensus problem in unreliable distributed systems (a brief survey). Yale University Tech Rep YALEU/DCS/RR-273
7. Fischer M, Lynch N (1982) A lower bound for the time to assure interactive consistency. Inf Proc Lett 14(4):183–186
8. Halpern J, Simons B, Strong R, Dolev D (1984) Fault-tolerant clock synchronization. Proc 3rd ACM Symp on Principles of Distributed Computing, pp 89–102
9. Lamport L, Melliar-Smith P (1985) Synchronizing clocks in the presence of faults. JACM 32(1):52–78
10. Lundelius J, Lynch N (1984) A new fault-tolerant algorithm for clock synchronization. Inf Control 62(2):190–204
11. Lamport L, Shostak R, Pease M (1982) The Byzantine generals problem. ACM Trans on Programming Languages and Systems 4(2):382–401
12. Mahaney S, Schneider F (1985) Inexact agreement: accuracy, precision and graceful degradation. Proc 4th ACM Symp on Principles of Distributed Computing, pp 237–249
13. Moses Y, Tuttle M (1988) Programming simultaneous actions using common knowledge. Algorithmica 3:121–169
14. Pease M, Shostak R, Lamport L (1980) Reaching agreement in the presence of faults. JACM 27(2):228–234