Asymptotically Optimal Algorithms For Approximate Agreement

A. D. Fekete

Department of Mathematics

Harvard University

Cambridge, MA 02138

fekete%h-ma1@harvard

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. In the failure-by-omission and Byzantine failure algorithms, each processor attempts to identify the faulty processors and corrects values transmitted by them to reduce the amount of disagreement. For each failure model and each value of S, we give a t-resilient algorithm using S rounds of communication which has convergence rate that is asymptotic to the best possible as the number of processors increases. If $S = t + 1$, exact agreement is obtained.

1 The Problem and Statement of Results

An important question in the design of fault-tolerent 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 ([F]). These problems have been studied in several models of computation, and it has been found that any solution resilient to t faulty processors requires $t + 1$ rounds of communication in the worst case ([PSL], [LSP], [FL]). In some practical situations complete agreement is not required - e.g. in synchronizing clocks ([LL]) 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 [DLPSW].

In this paper we study a t-resilient appoximate agreement problem in this form: there are n processors labelled $1, 2, \ldots, n$. 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 $f = |Fault|$. 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 condition: 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 $w(p) \in \rho\left(\hat{U}\right)$".

We will measure the performance of such an algorithm by the change in the range spanned by the values of the processors. Thus we measure performance in the crash-failure and failure-by-omission models by $K = \sup \dfrac{\delta(\{w(p) : p \in Corr\})}{\delta(U)}$ and in the Byzantine model by $K = \sup \dfrac{\delta(\{w(p) : p \in Corr\})}{\delta\left(\hat{U}\right)}$, 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 not known to the processors during the algorithm and in fact a given execution may be explained by more than one identification.

The greater generality of the Byzantine model means that any algorithm valid in that model is also valid with at least as good a performance in the other models. Similarly any lower bound on achievable values of K in the crash-failure model applies to the other models as well.

For the Byzantine model, the paper [DLPSW] gives an algorithm using only one round of communication, valid when $n > 3t$, with performance $K = \lceil (n-2t)/t \rceil^{-1}$. 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 present paper introduces an S-round algorithm valid when $n > 4t$, with performance

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

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

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

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

$$K \geq \frac{\sup(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ all } l_i \text{ nonnegative integers})}{(n+t)^S}$$

which is due to [DLPSW]. An interesting feature of our algorithm is that each processor tries to identify which of the other processors is faulty, and then ignores any information received from a known faulty processor to reduce the possibilities for disagreement.

In this paper we give a new lower bound for K in the crash-failure model, namely

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

for any algorithm using S rounds of communication. We also give an algorithm for the crash-failure model with performance

$$K \leq \frac{\sup(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ all } l_i \text{ nonnegative integers})}{(2n-2t)^S}$$

which is asymptotic to the optimum as n increases.

We offer an algorithm in the failure-by-omission model by combining parts of the algorithms from the other models. This has performance

$$K \leq \frac{\sup(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ all } l_i \text{ nonnegative integers})}{(2n-4t)^{S-1}(2n-2t)}$$

which is asymptotic to optimal.

It is worth noting that if $S = t+1$ the expression $\sup(l_1 \cdots l_S : l_1 + \cdots + l_S \leq t, \text{ all } l_i \text{ nonnegative integers})$ is zero, as one of the l_i must be zero, and so our algorithms give solutions to the exact agreement problem when run for $t+1$ rounds. In the Byzantine model this solution satisfies the strong validity condition that the value agreed on lies in the range of initial values of correct processors (this is not achieved by normal Byzantine agreement algorithm on each bit of the initial values unless some removal of extreme values is done). In each model our algorithm for S rounds starts by doing all the communication of the $S-1$ round algorithm, so it is possible to do

approximate agreement without knowing at the start how many rounds will be used. In fact, after each round the new values can be calculated as if that round were the last – this permits the values held by the correct processors to approach one another rapidly, finally agreeing if $t+1$ rounds are used.

The algorithms introduced here require exponential amounts of message traffic, like most other consensus or Byzantine Agreement algorithms. Coan has introduced a transformation which can encode algorithms of this type so as to require only polynomial communication ([C]). However in the Byzantine model 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 [DLPSW] (which involves only linear message traffic) will depend on the details of the system. In the crash-failure and failure-by-omission models Coan's transformation involves no overhead rounds and so is a definite improvement to our algorithms.

In §2 we give the notation and technical lemmas we will use later. §3 provides an intuitive introduction to the algorithms by discussing a similar but simpler algorithm and pointing out the modifications needed to get asymptotically optimal performance. §4 discusses the algorithm in the Byzantine model, and §5 gives the corresponding lower bound. In §6 and §7 we then give algorithm and lower bound for the crash-failure model, and §8 is devoted to the failure-by-omission model.

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 for detailed comments on a later draft, Leslie Lamport for suggestions about the crash-failure case and Yoram Moses for fruitful discussions about §7.

2 Notation and Lemmas

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

As in [DLPSW] we will try to reduce the range of values held by processors by using operators that act on multisets by removing extreme values. Let V be a multiset with $|V| = N$. We put $red_k(V)$ to be the multiset with $N-2k$ entries formed from V by removing the k highest entries and also the k lowest entries. We order the values by treating $\perp_r$ as greater than any real number and also as greater than $\perp_R$ if $r > R$. For the crash-failure or failure-by-omission models we will use similar operators $chop_k^r$ that prefer to remove $\perp_r$. 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 will ensure that a mean is only taken for multisets of real values. As examples:

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

- $center_3(\{-1,-1,0,1,\perp_1\}) = -0.5$
- $center_2(\{-1,-1,0,1,\perp_1\}) = -1/3$

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

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

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

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

Lemma 3 *Let V and W be multisets with $|V| = |W| = N$. Suppose that every entry in $V \cup W$ is one of v, w or $\perp_r$, and that $mult(\perp_r, V) \leq k$ and $mult(\perp_r, W) \leq k$. If $|mult(v,V) - mult(v,W)| + |mult(w,V) - mult(w,W)| \leq m$ then $|mult(v, chop_k^r(V)) - mult(v, chop_k^r(W))| \leq m$ and $|mult(w, chop_k^r(V)) - mult(w, chop_k^r(W))| \leq m$*

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

Lemma 4 *If V is a multiset with $|V| = N$, $mult(\perp_r, V) \leq k$ such that at least $N-k$ entries of V are different from $\perp_r$ and lie in the interval $[a,b]$, then every entry of $chop_{2k}^r(V)$ lies in $[a,b]$.*

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

Lemma 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 $mid_k(W)$ lies in $[a,b]$. By Lemma 2, the multisets $red_k(V)$ and $red_k(W)$ agree in at least $N-2k-m$ of their entries, and in each of the remaining m places, the entries can differ by at most $b-a$ as each lies in $[a,b]$. Thus $|mid_k(V) - mid_k(W)| = \frac{1}{N-2k}|\sum red_k(V) - \sum red_k(W)| \leq m(b-a)/(N-2k)$. Q.E.D.

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

Proof: The hypotheses show that in $double(V)$ there will be at most $2k$ entries that are not real, and all of them will be $\perp_1$ and so will be removed in forming $chop_k^1(V)$. Thus the resulting multiset has all its entries in $[a,b]$ and so its mean $center_k(V)$ also

lies in $[a,b]$. Similarly $center_k(W)$ also lies in $[a,b]$. Now as in the proof of Lemma 3 we observe that W is formed from V by at most m operations each replacing a value by $\perp_1$ or vice versa. So we need only prove that if V_1 and V_2 are multisets with $|V_1| = N$, $mult(\perp_1, V_1) \leq k-1$, $mult(\perp_r, V_1) = 0$ for $r > 0$, and every real entry of V_1 lies in the interval $[a,b]$ and such that V_2 is formed from V_1 by removing one occurrence of a value z and replacing it with $\perp_1$, then $|center_k(V_1) - center_k(V_2)| \leq (b-a)/(2N-2k)$. So we put $j = mult(\perp_r, V_1)$ and let Z denote the multiset of $2N-2j$ entries formed by removing all occurrences of $\perp_r$ from $double(V_1)$. Now $chop_k^r(V_1)$ is formed from Z by removing the $k-j$ highest entries and the $k-j$ lowest entries. On the other hand, $chop_k^r(V_2)$ is formed from Z by removing two occurrences of z and then removing the $k-j-1$ highest and $k-j-1$ lowest of the remaining entries. If z is among the $k-j-1$ lowest entries of Z, this is equivalent to removing the $k-j-1$ highest and $k-j+1$ lowest entries from Z. If z is among the $k-j-1$ highest entries of Z the net effect is to remove the $k-j+1$ highest and $k-j-1$ lowest entries from Z. Thus in these cases, we can obtain $chop_k^r(V_2)$ from $chop_k^r(V_1)$ either by removing an occurrence of the $k-j+1$ lowest entry of Z and adding an occurrence of the $k-j$ highest entry of Z, or else by replacing an occurence of the $k-j$ highest entry of Z by the $k-j$ lowest entry of Z. Clearly in these cases, the sum of the entries of $chop_k^1(V_1)$ differs from the sum of the entries of $chop_k^1(V_2)$ by the difference of two elements of the interval $[a,b]$ which is at most $b-a$. In the remaining case z lies between the $k-j$ lowest entry of Z (call it a') and the $k-j$ highest entry of Z (call it b'), but $chop_k^1(V_2)$ is obtained from $chop_k^1(V_1)$ by removing two occurences of z and replacing them with a' and b' which will alter the sum of the entries by $b'+a'-2z$ which is at most $b'-z$ (as $z \geq a'$) but this is bounded by $b-a$. Thus in every case

$$\begin{aligned} &|center_k(V_1) - center_k(V_2)| \\ &\quad = \tfrac{1}{2N-2k}|\textstyle\sum chop_k^1(V_1) - \sum chop_k^1(V_2)| \\ &\quad \leq (b-a)/(2N-2k) \end{aligned}$$

as required. Q.E.D.

3 Introduction to the Algorithms

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

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

- In round 1: Broadcast $v(p)$, and denote by $v(q_1,p)$ the value received by p from q_1 as $v(q_1)$. If the message from q_1 is missing set $v(q_1,p)$ to be $\perp_1$.
- In round 2: Broadcast the vector of values $\langle v(1,p), v(2,p),\ldots, v(n,p)\rangle$ and denote by $v(q_1,q_2,p)$ the value received by p from q_2 as $v(q_1,q_2)$. If the message from q_2 is missing set $v(q_1,q_2,p)$ to be $\perp_2$.
- In round r, for $r = 3,\ldots,S$, processor p will start with an array of n^{r-1} values $\langle v(q_1,q_2,\ldots,q_{r-1},p)$: each $q_i = 1,\ldots,n\rangle$. Now p should broadcast the array $\langle v(q_1,q_2,\ldots,q_{r-1},p)\rangle$. Denote by $v(q_1,\ldots,q_{r-1},q_r,p)$ the value received by p from q_r as $v(q_1,\ldots,q_{r-1},q_r)$. If the message from q_r is missing set $v(q_1,\ldots,q_{r-1},q_r,p)$ to be $\perp_r$.
- At the end of round S, processor p has an array of values $v(q_1,\ldots,q_S,p)$. Now let $W(q_1,\ldots,q_S,p)$ denote the multiset with a single entry $v(q_1,\ldots,q_S,p)$.
- For each r decreasing from $S-1$ to 1
  - for each choice of $q_1,\ldots,q_r$, form a multiset

$W(q_1,\ldots,q_r,*,\ldots,*,p)$
$= red_{(n-2t)^{S-r-1}t} \cup_{q_{r+1}=1}^{n} W(q_1,\ldots,q_r,q_{r+1},*,*,\ldots,*,p)$
where in every case the asterisks fill places so that there are $S+1$ entries, either asterisks or indices, to name each multiset.

- Now put $W(p) = \cup_{q_1=1}^{n} W(q_1,*,\ldots,*,p)$.
- Finally put $w(p) = mid_{(n-2t)^{S-1}t}(W(p))$.

The algorithm has two phases. First there are S rounds of communication, in each of which each active processor broad-

casts all the information it holds and collects the information sent to it. After round r processor p has an array of values $\langle v(q_1,\ldots,q_r,p) : \text{each } q_i = 1,\ldots,n\rangle$ where $v(q_1,\ldots,q_r,p)$ is the value p received from q_r representing the initial value $v(q_1)$ as transmitted by q_1 to q_2 in round 1, then relayed by q_2 to q_3 in round 2, and so on. In the second phase, after all communication has occurred, processor p builds for each choice of $q_1,\ldots,q_r$ a multiset $W(q_1,\ldots,q_r,*,\ldots,*,p)$ out of the collection of values $\{v(q_1,\ldots,q_r,q_{r+1},\ldots,q_S,p) : \text{each } q_j = 1,\ldots,n \text{ for } j > r\}$. Now if $q_r,q_{r+1},\ldots,q_S$ are all non-faulty then $v(q_1,\ldots,q_r,q_{r+1},\ldots,q_S,p) = v(q_1,\ldots,q_r)$ and in fact the method of constructing $W(q_1,\ldots,q_r,*,\ldots,*,p)$ by successively combining multisets and removing extreme values is designed to ensure that $W(q_1,\ldots,q_r,*,\ldots,*,p)$ is a multiset of size $(n-2t)^{S-r}$ which is a good representative for $v(q_1,\ldots,q_r)$ in that

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

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

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

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

which shows that

$$K \leq \frac{\sup\{l_1 l_2\cdots l_S : l_1+\cdots l_S \leq t,\ \text{all } l_i \text{ nonnegative integers}\}}{(n-2t)^S}$$

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

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

The algorithms of §6 and §8 also obtain better performance than the basic algorithm above by using the operators $chop_k^r$ and $center_k$ which are more complicated than red_k and mid_k but are specially adapted to the situations where the only differences between multisets $W(q_1,\ldots,q_r,*,\ldots,*,p_0)$ and $W(q_1,\ldots,q_r,*,\ldots,*,p_1)$ are due to replacing a value by $\perp_r$ (unlike the Byzantine case where one value can be replaced by another).

4 The Byzantine Model: The Algorithm

An overview — During each round of communication a correct processor p broadcasts information it holds in the array $\tilde{v}(p_1,\ldots,p_{r-1},p)$, collects the information sent to it in an array $v(p_1,\ldots,p_r,p)$, tries to deduce which processors are faulty, and then modifies the information it received from processors known to be faulty to form the new array $\tilde{v}(p_1,\ldots,p_r,p)$. The only method a correct processor p uses to detect that process q is faulty, is to examine the n values which reach p representing some information that was broadcast by q and then relayed to p by each recipient. If q were correct then every processor would have received the same value in the broadcast and then the correct processors (at least $n-t$ of them) would all have sent the same value to p. Thus if p finds fewer than $n-t$ values the same

among the n it received, it can deduce that q was faulty. After the S rounds of communication, a correct processor will have an array of n^S values to operate on. In S steps this array is used to form a collection of $(n-2t)(n-4t)^{S-1}$ values by repeatedly removing extreme values from subcollections and then combining subcollections. Finally this collection of values is averaged to give the processor's new value.

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

- Set $\tilde{v}(p) = v(p)$.
- In round 1:
  - Broadcast $\tilde{v}(p)$, and denote by $v(q_1, p)$ the value received by p from q_1 purporting to be $\tilde{v}(q_1)$. If the mesage from q_1 is missing or malformed set $v(q_1, p)$ to be $\perp_1$.
  - Set $Fault(p, 1)$ to be the empty set.
  - Set $\tilde{v}(q_1, p) = v(q_1, p)$.
- In round 2:
  - Broadcast the vector of values $\langle \tilde{v}(1,p), \tilde{v}(2,p), \ldots, \tilde{v}(n,p) \rangle$ and denote by $v(q_1, q_2, p)$ the value received by p from q_2 purporting to be $\tilde{v}(q_1, q_2)$. If the message from q_2 is missing or malformed set $v(q_1, q_2, p)$ to be $\perp_2$.
  - For each q_1 consider the multiset $\{v(q_1, 1, p), v(q_1, 2, p), \ldots, v(q_1, n, p)\}$; if the most frequently occurring element in this multiset has multiplicity less than $n - t$ say that "q_1 has been detected as faulty by p in round 2".
  - Set $Fault(p, 2)$ to be the set of q which have been detected as faulty by p in round 2.
  - Set $\tilde{v}(q_1, q_2, p) = \begin{cases} v(q_1, q_2, p) & \text{if } q_2 \notin Fault(p,2) \\ \perp_2 & \text{if } q_2 \in Fault(p,2) \end{cases}$.
- In round r, for $r = 3, \ldots, S$, processor p will start with an array of n^{r-1} values $\langle \tilde{v}(q_1, q_2, \ldots, q_{r-1}, p) : \text{each } q_i = 1, \ldots, n \rangle$ and a set $Fault(p, r-1)$ of processors already detected as faulty by p. Now p should
  - Broadcast the array $\langle \tilde{v}(q_1, q_2, \ldots, q_{r-1}, p) \rangle$.
  - Denote by $v(q_1, \ldots, q_{r-1}, q_r, p)$ the value received by p from q_r purporting to be $\tilde{v}(q_1, \ldots, q_{r-1}, q_r)$. If the message from q_r is missing or malformed set $v(q_1, \ldots, q_r, p)$ to be $\perp_r$.
  - For every choice of indices $q_1, \ldots, q_{r-1}$, consider the multiset $\{v(q_1, \ldots, q_{r-1}, 1, p), v(q_1, \ldots, q_{r-1}, 2, p), \ldots, v(q_1, \ldots, q_{r-1}, n, p)\}$. If the most frequently occurring element has multiplicity less than $n - t$, say that "q_{r-1} has been detected as faulty by p in round r" (Note that 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 $\tilde{v}(q_1, \ldots, q_{r-1}, q_r, p) = \begin{cases} v(q_1, \ldots, q_{r-1}, q_r, p) \\ \quad \text{if } q_r \notin Fault(p, r) \\ \perp_r \text{ if } q_r \in Fault(p, r) \end{cases}$
  - Now p is ready to start round $r + 1$.
- 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 $q_1, \ldots, q_r$, form a multiset

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

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

The behaviour of the algorithm is explained by the following:

Theorem 1 *In the algorithm above as a convention we set* $Fault(p, 0) = \emptyset$, $Fault(p, S+1) = \{1, \ldots, n\} \setminus Corr$. *We put* $Exposed(r) = \cap_{p \in Corr} Fault(p, r)$ *and* $l_r = |Exposed(r+1)| - |Exposed(r)| = |Exposed(r+1) \setminus Exposed(r)|$ *Then we can conclude:*

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

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

$$W(q_1,\ldots,q_r,*,\ldots,*,p_0) = W(q_1,\ldots,q_r,*,\ldots,*,p_1)$$

(iii): *If* $q_r \in Exposed(r)$ *and* $p_0 \in Corr$, $p_1 \in Corr$ *then*

$$W(q_1,\ldots,q_r,*,\ldots,*,p_0) = W(q_1,\ldots,q_r,*,\ldots,*,p_1)$$

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

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

Proof: First we observe that if $p \in Corr$, $q \in Corr$ then $q \notin Fault(p,r)$ This is proved 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 $q \notin Fault(q_r,1)$ 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 of values $\{v(q_1,\ldots,q_{r-2},q,1,p), v(q_1,\ldots,q_{r-2},q,2,p),\ldots, v(q_1,\ldots,q_{r-2},q,n,p)\}$ contains at least $(n-t)$ entries each of which is $\tilde{v}(q_1,\ldots,q_{r-2},q)$. So q is not detected as faulty by p in round r, but by the induction hypothesis $q \notin Fault(p,r-1)$ so we see $q \notin Fault(p,r)$ as required. Now the theorem follows easily by descending induction on r, using the lemmas of §2 and the observations that if $q_r \in Exposed(r)$ then for any correct q_{r+1} we have $\tilde{v}(q_1,\ldots,q_r,q_{r+1}) = \perp_r$ while if $q_r \notin Exposed(r+1)$ then for fixed choice of $q_1,\ldots,q_{r-1}$ at least $n-2t$ of the correct processors q_{r+1} have the same value for $\tilde{v}(q_1,\ldots,q_{r-1},q_r,q_{r+1})$.

When we apply Theorem 1 with $r = 1$ we obtain

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

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

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

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

Notice that (iii) tells us nothing as $Exposed(1) = \emptyset$. Now if $p \in Corr$ we see that $\cup_{q_1=1}^{n} W(q_1,*,\ldots,*,p)$ contains at least $(n-t)(n-4t)^{S-1}$ entries in the range $\rho(\hat{U})$ 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 $p_0 \in Corr$, $p_1 \in Corr$. Then

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

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

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

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

$$K \leq \frac{\sup\{l_1 l_2\cdots l_S : l_1 + \cdots l_S \leq t,\ \text{all } l_i \text{ nonnegative integers}\}}{(n-4t)^{S-1}(n-2t)}$$

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

5 The Byzantine Model: A Lower Bound

This section gives a formal account of a lower bound, due to [DLPSW], on achievable performance for any S-round approximate agreement algorithm in the Byzantine model. Any algorithm for solving the S-round approximate agreement problem can be given in the following standard form, 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 $u(q_1)$.
- 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 $u(q_1,\ldots,q_r)$.

- Finally a processor $p \in Corr$ applies a function f to its *view*, the array $\langle u(q_1,\ldots,q_S,p)\rangle$ of n^S values, to produce its new value $w(p)$.

Different algorithms are given by different choices of the function f. Notice that the algorithm of §4, which involves computing and modifying values between rounds of communication, is equivalent to one in the standard form because all the computation and modification can be simulated by each processor after all the information is exchanged. So suppose we are given a function f for which the algorithm satisfies the validity condition. Let $l_1,l_2,\ldots,l_S$ be any positive integers so that $l_1+\cdots+l_S \le t$. 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 'alphabetically', that is $(i_1,\ldots,i_S) < (j_1,\ldots,j_S)$ if there is some r so that (i) $i_k \le j_k$ 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

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

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

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

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

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

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

to which our algorithm is asymptotic as n increases.

The reader can verify that the following construction gives an execution as required with $M_{(i_1,i_2,\ldots,i_S)}$ as the view for p_0, and $M_{(i_1,\ldots,i_S)++}$ as the view for p_1 : The faulty processors are those p such that there is an r with $\lceil p/l_r \rceil = i_r$. Since for each r at most l_r values of p satisfy this condition, the total number of faulty processors is at most $l_1+\ldots+l_S \le t$. Choose p_0 and p_1 from among the correct processors. Let $v(p)$ be 1 if $\lceil p/l_1 \rceil \le i_1$, 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 $\lceil q/l_2 \rceil \le i_2$, 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 where $\lceil p/l_r \rceil = i_r$ broadcast the array $\langle u(q_1, \ldots, q_{r-1}, p) \rangle$. The remaining p form another array with

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

  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 $\lceil p/l_S \rceil = i_S$ broadcast the array $\langle u(q_1, \ldots, q_{S-1}, p) \rangle$. The remaining p form another array with

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

  These p send the array $\langle u(q_1, \ldots, q_{S-1}, p) \rangle$ to those q where $q \neq p_0$, but to p_0 they send instead the array $\langle u'(q_1, \ldots, q_{S-1}, p) \rangle$.

  – 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.

In fact, the lower bound just derived can be improved slightly by finding other multi-indices I for which M_I is the view in some execution with all correct processors having 0 as initial value, and by finding multi-indices I for which an execution exists in which one correct processor receives M_I as view and another receives $M_{(I++)++}$ as view.

6 The Crash-Failure Model: The Algorithm

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

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

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

- In round 2: Broadcast the vector of values $\langle v(1,p), v(2,p), \ldots, v(n,p) \rangle$ and denote by $v(q_1, q_2, p)$ the value received by p from q_2 as $v(q_1, q_2)$. If the message from q_2 is missing set $v(q_1, q_2, p)$ to be $\perp_2$.

- In round r, for $r = 3, \ldots, S$, processor p will start with an array of n^{r-1} values $\langle v(q_1, q_2, \ldots, q_{r-1}, p) : \text{each } q_i = 1, \ldots, n \rangle$. Now p should broadcast the array $\langle v(q_1, q_2, \ldots, q_{r-1}, p) \rangle$. Denote by $v(q_1, \ldots, q_{r-1}, q_r, p)$ the value received by p from q_r as $v(q_1, \ldots, q_{r-1}, q_r)$. If the message from q_r is missing set $v(q_1, \ldots, q_{r-1}, q_r, p)$ to be $\perp_r$.

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

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

  – for each choice of $q_1, \ldots, q_r$, form a multiset

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

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

- Now put $W(p) = \cup_{q_1=1}^{n} W(q_1, *, \ldots, *, p)$.
- Finally put $w(p) = center_{(2n-2t)^{S-1}_t}(W(p))$.

The behaviour of the algorithm is explained by the following:

Theorem 2 *In the algorithm above, for each $r = 1, \ldots, S$ let $Fail(r)$ denote the set of processors that have failed before sending any of the messages in round r. Also as a convention we set $Fail(S+1) = \{1, \ldots, n\} \setminus Corr$. We put $l_r = |Fail(r+1)| - |Fail(r)| = |Fail(r+1) \setminus Fail(r)|$ Then we can conclude:*

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

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

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

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

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

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

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

Proof: First we observe that if $p \in Corr$, then $p \notin Fail(r)$ for $r = 1, \ldots, S+1$. Now the theorem follows easily by descending induction on r, using the lemmas of §2 and the observations that if $q_r \notin Fail(r+1)$ then q_r sent all its messages in round r, so that every q_{r+1} that has not failed before starting round $r+1$ has $v(q_1, \ldots, q_r, q_{r+1}) = v(q_1, \ldots, q_r)$, while on the other hand if $q_r \in Fail(r)$ then q_r sent no messages in round r, so that every q_{r+1} that has not failed before starting round $r+1$ has $v(q_1, \ldots, q_r, q_{r+1}) = \perp_r$.

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

$$\begin{aligned} &|mult(v(q_1), W(q_1, *, \ldots, *, p_0) - mult(v(q_1), W(q_1, *, \ldots, *, p_1)| \\ &\quad \leq l_2 \cdot l_3 \cdots l_S \end{aligned}$$

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

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

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

$$K \leq \frac{\sup\{l_1 l_2 \cdots l_S : l_1 + \cdots l_S \leq t, \text{ all } l_i \text{ nonnegative integers}\}}{(2n-2t)^S}$$

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

7 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 standard form as in §5, where all information is exchanged for S rounds giving each processor p a view $\langle v(q_1, \ldots, q_S, p)\rangle$ and then p applies a function f to the view to give its new value $w(p)$.

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

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

Lemma 7 *Let $1 \leq r \leq S-1$. Let $\rho = \rho_0$ be an execution such that no failures occur after the end of round r, and the number of failures before the start of round i is at most $l_1+\ldots+l_i$ for any i. Denote by $\hat{\rho}$ the execution which is identical to ρ for the first $r-1$ rounds but has no failures during any later round. Then $\rho \sim^{N(r)} \hat{\rho}$ where $N(r) = \sum_{i=r+1}^{S} \prod_{j=i}^{S} 2m_j + 1$.*

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

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

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

processors $(k-1)l_1+1,\ldots,kl_1$ fail in round 1 before sending any messages, but no other failures occur. Let ρ_{3k} denote the execution where processors $1,\ldots,kl_1$ have initial value 1, and the others have initial value 0, and where no failures occur. By Lemma 7, $\rho_{3(k-1)} \sim^{N(1)} \rho_{3k-2}$ and $\rho_{3k-1} \sim^{N(1)} \rho_{3k}$. Also the view of every processor is the same in ρ_{3k-2} as in ρ_{3k-1} since the initial value of a processor that fails before sending any message is irrelevant, and so $\rho_{3k-2} \approx \rho_{3k-1}$. Since $\rho_{3m_1} = \hat{\rho}$, we have $\rho \sim^N \hat{\rho}$ where $N = 2m_1(N(1)+1) \leq \prod_{j=1}^{S} 2m_j + 2$ as we see by writing $2m_j+2$ as $(2m_j+1)+1$ and expanding the product. In the case $S=1$ for each $k=1,\ldots,m_1$ let p_k and q_k be the least and greatest processor indices that are not in the range $(k-2)l_1+1,\ldots,kl_1$. 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 $1,\ldots,p_k$. 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 $1,\ldots,p_k$. The view of p_k is the same in $\rho_{2(k-1)}$ as in ρ_{2k-1} so $\rho_{2(k-1)} \approx \rho_{2k-1}$. Similarly the view of q_k is the same in ρ_{2k-1} as in ρ_{2k} so $\rho_{2k-1} \approx \rho_{2k}$. As the view of p_{m_1} is the same in ρ_{2m_1} as in $\hat{\rho}$ we have that $\rho_{2m_1} \approx \hat{\rho}$, and so $\rho \sim^N \hat{\rho}$, where $N = 2m_1+1 \leq 2m_1+2$.

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

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

In fact by paying closer attention to the cases when successive executions look the same to all processors, we can improve the denominator to $(2n+2t)^S$.

8 The Failure-By-Omission Model

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

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

-

- Set $\tilde{v}(p) = v(p)$.
- In round 1:
  - Broadcast $\tilde{v}(p)$, and denote by $v(q_1,p)$ the value received by p from q_1 as $\tilde{v}(q_1)$. If the message from q_1 is missing set $v(q_1,p)$ to be $\perp_1$.
  - Set $Fault(p,1)$ to be the empty set.
  - Set $\tilde{v}(q_1,p) = v(q_1,p)$.
- In round 2:
  - Broadcast the vector of values $\langle \tilde{v}(1,p), \tilde{v}(2,p),\ldots, \tilde{v}(n,p)\rangle$ and denote by $v(q_1,q_2,p)$ the value received by p from q_2 as $\tilde{v}(q_1,q_2)$. If the message from q_2 is missing set $v(q_1,q_2,p)$ to be $\perp_2$.
  - For each q_1 consider the multiset $\{v(q_1,1,p), v(q_1,2,p), \ldots, v(q_1,n,p)\}$; if any entry is $\perp_1$ say that "q_1 has been detected as faulty by p in round 2".
  - Set $Fault(p,2)$ to be the set of q which have been detected as faulty by p in round 2.

– Set $\tilde{v}(q_1,q_2,p) = \begin{cases} v(q_1,q_2,p) & \text{if } q_2 \notin Fault(p,2) \\ \perp_2 & \text{if } q_2 \in Fault(p,2) \end{cases}$.

- In round r, for $r = 3,\ldots,S$, processor p will start with an array of n^{r-1} values $\langle \tilde{v}(q_1,q_2,\ldots,q_{r-1},p) : \text{each } q_i = 1,\ldots,n\rangle$ and a set $Fault(p,r-1)$ of processors already detected as faulty by p. Now p should
  - Broadcast the array $\langle \tilde{v}(q_1,q_2,\ldots,q_{r-1},p)\rangle$.
  - Denote by $v(q_1,\ldots,q_{r-1},q_r,p)$ the value received by p from q_r as $\tilde{v}(q_1,\ldots,q_{r-1},q_r)$. If the message from q_r is missing set $v(q_1,\ldots,q_{r-1},q_r,p)$ to be $\perp_r$.
  - For every choice of indices $q_1,\ldots,q_{r-1}$, consider the multiset $\{v(q_1,\ldots,q_{r-1},1,p), v(q_1,\ldots,q_{r-1},2,p),\ldots, v(q_1,\ldots,q_{r-1},n,p)\}$. If any entry is $\perp_{r-1}$ say that "q_{r-1} has been detected as faulty by p in round r" (Note that several choices of $q_1,\ldots,q_{r-2}$ may lead to the same q_{r-1} being detected).
  - Set $Fault(p,r) = Fault(p,r-1) \cup \{q : q \text{ has been detected as faulty by } p \text{ in round } r\}$.
  - Set $\tilde{v}(q_1,\ldots,q_{r-1},q_r,p) = \begin{cases} v(q_1,\ldots,q_{r-1},q_r,p) \\ \quad \text{if } q_r \notin Fault(p,r) \\ \perp_r \text{ if } q_r \in Fault(p,r) \end{cases}$.
  - Now p is ready to start round $r+1$.
- 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 $q_1,\ldots,q_r$, form a multiset

$W(q_1,\ldots,q_r,*,\ldots,*,p)$
$= chop^{r+1}_{(2n-4t)^{S-r-1}2t} \cup_{q_{r+1}=1}^{n} W(q_1,\ldots,q_r,q_{r+1},*,*,\ldots,*,p)$
where in every case the asterisks fill places so that there are $S+1$ entries, either asterisks or indices, to name each multiset.
- Now put $W(p) = \cup_{q_1=1}^{n} W(q_1,*,\ldots,*,p)$.
- Finally put $w(p) = center_{(2n-4t)^{S-1}t}(W(p))$. (Note that the amount of reduction is different from that in previous steps).

The behaviour of the algorithm is explained by the following:

Theorem 3 *In the algorithm above, for each $r = 1,\ldots,S$ let $Fail(r)$ denote the set of processors that have failed before sending any of the messages in round r. Let $Exposed(r) = Fail(r) \cup \cap_{p\notin Fail(r+1)} Fault(p,r)$. Also as a convention we set $Exposed(S+1) = \{1,\ldots,n\}\setminus Corr$. Note that $Exposed(r) \subset Exposed(r+1)$. We put $l_r = |Exposed(r+1)| - |Exposed(r)| = |Exposed(r+1)\setminus Exposed(r)|$ Then we can conclude:*

(i): *If $p \in Corr$ 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 $p \in Corr$ 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 $p \in Corr$ then*

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

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

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

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

We have by (ii) and (iii) 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

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

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

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

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

$$K \leq \frac{\sup\{l_1 l_2 \cdots l_S : l_1 + \cdots l_S \leq t, \text{ all } l_i \text{ nonnegative integers}\}}{(2n-4t)^{S-1}(2n-2t)}$$

[C] B. Coan, "A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols", Proceedings of the 5th ACM Symposium on Principles of Distributed Computing, August 1986.

[CD] B. Coan, C. Dwork, "Simultaneity is Harder than Agreement", Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems, 141–150, January 1986.

[D] D. Dolev, "The Byzantine Generals Strike Again", Journal of Algorithms 3, 14–30 (1982).

[DLPSW] D. Dolev, N. Lynch, S. Pinter, E. Stark, W. Weihl, "Reaching Approximate Agreement in the Presence of Faults", to appear in JACM.

[DM] C. Dwork, Y. Moses, "Knowledge and Common Knowledge in a Byzantine Environment I: Crash Failures", Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge, 149–169, March 1986.

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

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

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

[LSP] L. Lamport, R. Shostak, M. Pease, "The Byzantine Generals Problem", ACM Transactions on Programming Languages and Systems 4, 2, 382–401 (1982).

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