ON THE MESSAGE COMPLEXITY OF BINARY BYZANTINE AGREEMENT UNDER CRASH FAILURES

Eugene S. Amdur
Samuel M. Weber
Vassos Hadzilacos

Department of Computer Science
University of Toronto, Toronto, Canada

ABSTRACT

The binary Byzantine Agreement problem requires $n-1$ receivers to agree on the binary value broadcast by a sender even when some of these n processes may be faulty. We investigate the message complexity of protocols that solve this problem in the case of crash failures. In particular, we derive matching upper and lower bounds on the total, worst and average case number of messages needed in the failure-free executions of such protocols.

More specifically, we prove that any protocol that tolerates up to t faulty processes requires a total of at least $n+t-1$ messages in its failure-free executions — and, therefore, at least $\lceil (n+t-1)/2 \rceil$ messages in the worst case and $\min(P_0, P_1) \cdot (n+t-1)$ messages in the average case, where P_v is the probability that the value of the bit that the sender wants to broadcast is v. We also give protocols that solve the problem using only the minimum number of messages for these three complexity measures. These protocols can be implemented by using 1-bit messages. Since a lower bound on the number of messages is also a lower bound on the number of message bits, this means that the above tight bounds on the number of messages are also tight bounds on the number of message bits.

September 5, 1991

1. Introduction

Byzantine Agreement (BA) is a fundamental problem in fault tolerance. Roughly speaking it can be formulated as follows. There is a collection of n processes consisting of one *sender* and $n-1$ *receivers*. The processes can communicate only by exchanging messages. Initially, the sender has a value drawn from a set of at least two decision values. In this paper we consider the *binary* BA problem, in which there are exactly two decision values, 0 and 1. Associated with each process is an algorithm prescribing its behaviour — in particular, the messages it should send. A process is correct in a particular execution if its behaviour in that execution is that prescribed by its algorithm; otherwise it is faulty. The objective is to design a protocol, i.e. an algorithm for each process, so that:

1. every correct process eventually chooses a decision value;
2. no two correct processes choose different decision values; and
3. if the sender is correct then all correct processes choose the sender's initial value.

In designing such a protocol an upper bound $t < n$ on the number of possible faulty processes is assumed. The protocol is required to guarantee these three properties only in executions in which the number of faulty processes does not exceed t.

The problem was introduced, albeit in a slightly different form, by Pease *et al.* [17] and has subsequently been subjected to voluminous investigation. Fischer [11] provides an excellent, though by now somewhat dated, survey of models and results concerning the problem.

Among the many types of process failures under which the BA problem has been studied the simplest, and possibly of greatest practical value, is the class of *crash failures.* This means that the only way in which a process' behaviour can deviate from that specified by its algorithm is to simply suspend all activity — it no longer sends any messages and it will not reach a decision, unless it has already done so.

In this paper we investigate the message complexity of crash failure tolerant binary BA protocols in the failure-free executions. The motivation for studying message complexity is that the number of messages used by a protocol is an important, possibly the most important, factor that determines its performance [13]. The motivation for restricting our investigation to failure-free executions is that it seems plausible that the overwhelming majority of executions in a real-life system are failure-free. If so, it is reasonable to design protocols optimised for this most frequent case. Such protocols will be fault-tolerant, but they incur the cost of anticipating faults in the — hopefully — rare instances when faults do, in fact, occur. Most previous work on lower bounds for BA has concentrated on complexity measures other than the number of messages — especially on the so-called "round complexity", a measure of parallel time [12, 7, 8, 16]. Other papers which address the message complexity of BA protocols include [6, 2, 4]. These study the (worst case) message complexity over *all* executions, not just the failure-free ones. Furthermore, they are concerned with more liberal models of process failures than crash failures.

There are various reasonable message complexity measures that one might wish to consider:

Total: The sum of the number of messages in the failure-free executions.

Worst case: The maximum of the number of messages in the failure-free executions.

Average case: The expected number of messages in the failure-free executions, given the probabilities P_0 and P_1 that the sender's initial value is 0 and 1, respectively.

In this paper we prove that for the total complexity, $n+t-1$ messages are necessary and sufficient; for the worst case complexity $\lceil (n+t-1)/2 \rceil$ messages are necessary and sufficient; and for the average case complexity $\min(P_0, P_1) \cdot (n+t-1)$ messages are necessary and sufficient. As far as the lower bounds are concerned, the hard part is getting the result for the total complexity; the other two follow immediately. For the upper bounds, we give two protocols that match the lower bounds for the worst case and average case complexity respectively. Both protocols match the lower bound for the total complexity.

All three of the complexity measures mentioned above count only the number of messages, placing no restrictions to the size of messages. In practice we cannot allow messages to be arbitrarily long, so a complexity measure which is oblivious to the size of messages may be too crude. A more refined measure would be to actually count the number of bits contained in the messages. We shall call this the *bit complexity*. (It can be any one of the three types: total, worst case or average case.) Since a message must convey at least one bit, a lower bound on the message complexity is also a lower bound on the bit complexity. Fortuitously, our two protocols which achieve the optimal worst and average case message complexities can be implemented by using just one bit per message! Thus they provide upper bounds not only on the message complexities but on the bit complexities as well. Our results then imply matching bounds on the total, worst case and average case bit complexities.

The rest of the paper is organised as follows: In Section 2 we describe our model of computation and make more precise various concepts and terms mentioned previously. In Section 3 we look at the problem under the (extremely) simplifying assumption that there are no faulty processes at all ($t = 0$); this sheds some light into the problems encountered and techniques used later on. Section 4 contains the lower bound and Section 5 the upper bound results. We conclude in Section 6 with some open problems.

2. The model

In this section we describe the model of computation we use. Consider a set of n processes containing a distinguished *sender* and $n-1$ *receivers*. The sender has a local variable *init* which is initialised to one of the decision values (0 or 1, in our case). Every process has an initially undefined, write-once local variable *decision*. The value that the process assigns to that variable is its decision value. The processes can communicate only by exchanging messages over a completely connected, reliable network.

Computation by the processes proceeds in synchronous *rounds*. In each round each process takes one step, consisting of the following actions, performed in the specified order:

1. Send messages to a subset of the processes.
2. Receive all messages sent to it by other processes in *that* round.
3. On the basis of the messages received in all rounds including the present one, possibly assign a decision value to variable *decision*.

Notice how (2) embodies the assumption that the communication network is reliable.

Let Π denote the set of processes. Let M be the set consisting of the messages that the processes can send to each other and the symbol λ, used to denote the lack of a message. Let $V = \{0, 1, \mathbf{undefined}\}$ be the set of possible values of the *decision* variables. Formally, a round is defined as a pair (S, D), where $S : \Pi \times \Pi \to M$ and $D : \Pi \to V$. $S(p, q)$ is the message (possibly λ) sent by p to q in that round; and $D(p)$ is the value of process p's *decision* variable at the end of that round.

A *run* E is a finite sequence

$$\mathbf{init}, (S_1, D_1), (S_2, D_2), \ldots (S_k, D_k) \tag{$*$}$$

where **init** is either 0 or 1, and each (S_i, D_i) is a round. This represents the execution in which the sender's initial value is **init** and then rounds occur as described by the pairs (S_i, D_i). If $0 \leq j \leq k$, E^j is the prefix of E up to and including round j; E^0 consists of just **init**. Note that E^j is also a run.

The *view* of a process p in a run E, denoted $\mathbf{view}(p, E)$, is the sequence of messages sent to p in each of the rounds of E. In the special case where p is the sender, we include **init** in the view of p. More precisely, if E is the run (*):

$$\mathbf{view}(p, E) = \begin{cases} \mathbf{init}, S_1|(\Pi \times \{p\}), S_2|(\Pi \times \{p\}), \ldots, S_k|(\Pi \times \{p\}), & \text{if } p \text{ is the sender} \\ S_1|(\Pi \times \{p\}), S_2|(\Pi \times \{p\}), \ldots, S_k|(\Pi \times \{p\}), & \text{otherwise} \end{cases}$$

where $f|A$ denotes the restriction of function f to domain A.

Let $\mathbf{Views}_p$ be the set of views of process p. An *algorithm* for p specifies what messages p should send in each round and what, if anything, it should decide. More formally, we define an algorithm for p to be a pair (Σ_p, Δ_p) where

$$\Sigma_p : \mathbf{Views}_p \times \Pi \to M$$

specifies the message (possibly λ) that p should send to each process, given its view; and

$$\Delta_p : \mathbf{Views}_p \to V$$

specifies p's decision given its view. Naturally, we require that if we have two views $v, v' \in \mathbf{Views}_p$ where v is a prefix of v' and $\Delta_p(v) \neq \mathbf{undefined}$ then $\Delta_p(v') = \Delta_p(v)$. In other words, decisions are irrevocable; this formalises that *decision* is a write-once variable.

A *protocol* is a collection of algorithms, one for each process.

Relative to a fixed protocol, we say that p is *correct* in run $E = \mathbf{init}, (S_1, D_1), \ldots, (S_k, D_k)$ if for each $1 \leq i \leq k$,

$$\text{for all } q \in \Pi,\ S_i(p, q) = \Sigma_p(\mathbf{view}(p, E^{i-1}), q) \quad \text{and}$$
$$D_i(p) = \Delta_p(\mathbf{view}(p, E^i)).$$

That is, the messages sent by p and the decision reached by it are in accordance with its algorithm in every round.

A run E is *failure-free* if every process is correct in E.

In this paper we are interested in the BA problem when faulty processes fail by crashing. A process crashes if it stops before its algorithm stipulates that it should. A crash can interrupt a process' actions at any point, except that we are assuming that the sending of a single message is an atomic action, so that a process cannot crash having sent only part of a message. However, it is perfectly possible for a process to crash when it has sent only a subset of the messages it is supposed to send in a round. More precisely, p *crashes in round i of run E* if:

1. p is correct in E^{i-1} but not in E^i;
2. for all $q \in \Pi$, if $S_i(p,q) \neq \Sigma_p(\mathbf{view}(p, E^{i-1}), q)$ then $S_i(p,q) = \lambda$; and
3. for all $j > i$ and all $q \in \Pi$, $S_j(p,q) = \lambda$.

From now on, when we say that a process is *faulty* in a run E, we mean that it crashes in some round of E. We are interested in runs in which at most t processes are faulty, where t is a fixed parameter. We say that a protocol is a *t-resilient protocol for BA* if there is some $k \in \mathrm{N}$ such that for every k-round run $E = \mathbf{init}, (S_1, D_1), \ldots, (S_k, D_k)$ with at most t faulty processes:

1. [Termination] if p is correct in E then $D_k(p) \neq$ **undefined**,
2. [Agreement] if p, q are correct in E then $D_k(p) = D_k(q)$, and
3. [Validity] if the sender is correct in E then for any p which is correct in E, $D_k(p) = \mathbf{init}$.

Note that in our definition of algorithm we stated that Σ_p and Δ_p are *functions*, as opposed to relations. In this way we are implicitly assuming that processes act deterministically. This may appear as a dubious assumption since it is known that with respect to certain complexity measures, such as worst case number of rounds over all runs, randomised protocols have significant advantages over deterministic ones [9, 10]. However, it can be shown that with respect to the complexity measures of interest in this paper — i.e., various types of message complexity *in the failure-free runs* — randomisation is of no help [15, Appendix A]. Thus, we can make this assumption with no loss of generality.

Since we assume that processes are deterministic, a failure-free run is completely determined by the sender's initial value. Thus, in the case of binary BA, a protocol has exactly two failure-free runs, denoted H_0 and H_1, where the subscript indicates the sender's initial value.

3. A simple special case

Let us begin by looking at the problem under the assumption that there are no faulty processes, i.e., when $t = 0$. Though this is an immense simplification, it will provide insights that will be useful to keep in mind when we study the general case.

Probably the most natural protocol in this case is for the sender to broadcast its value to all receivers in the first round; a receiver simply decides the value it receives. It is possible to improve on this. It is a well-known trick that, in synchronous systems, the *absence* of a message can be used to convey information. (For example, Cook *et al.* [5] make use of this trick.) We can exploit this idea as follows. Each receiver adopts the convention that the lack of a message from the sender means that it should decide one of the two values. It is not necessary that the same convention be adopted by all receivers but we shall assume that the sender knows which convention each receiver uses. Let R_v be the set of receivers which interpret the lack of a message in round 1 to mean that they should decide v, where $v \in \{0, 1\}$. In the improved protocol, if the sender's initial value is v, the sender need only send a message to the processes in $R_{\overline{v}}$.[1]

No matter how we distribute the receivers between R_0 and R_1, the total message complexity of this protocol is $n-1$, since each receiver gets a message in exactly one of the two runs. It is quite easy to see that this is the best we can do. Each receiver *must* get at least one message in one of the two runs. For, a receiver that gets no message in both runs of the protocol has the same view in both runs and must decide the same value in both, in violation of the Validity condition. This implies a lower bound of at least $n-1$ messages for the total complexity.

If we wish to minimise the worst case number of messages then the $n-1$ receivers should be distributed equally between R_0 and R_1, so that one of the two sets has $\lfloor (n-1)/2 \rfloor$ and the other has $\lceil (n-1)/2 \rceil$ receivers. With this choice for R_0 and R_1, the worst case message complexity of the protocol is $\lceil (n-1)/2 \rceil$. It is easy to see that this is the best we can do.

If we wish to minimise the expected number of messages, then all processes should adopt the convention that when they do not get a message from the sender they should decide the value which is more likely to be the initial value of the sender. In other words, if $P_v > P_{\overline{v}}$, R_v has all the receivers and $R_{\overline{v}} = \emptyset$. With this choice for R_0 and R_1, the average case message complexity of the protocol is $\min(P_0, P_1) \cdot (n-1)$. It is easy to see that this is the best we can do.

The problem with this simple protocol when we allow failures is that the lack of a message no longer has a single meaning. A correct receiver in R_v may fail to get a message either because the sender's initial value is v *or* because the sender is faulty. It is crucial for a receiver to remove this uncertainty before it decides — otherwise agreement may be compromised. Removing this uncertainty requires additional communication. Our lower bound result essentially shows that there has to be a set B of enough receivers (namely, at least t of them) which can distinguish the two failure-free runs by more than the fact that they receive a message in one but not in the other; they must either receive a message in both failure-free runs, or they must receive more than one message in one of them. Our upper bounds are embellishments of the simple protocol given above where the processes in B use their superior ability to distinguish the failure-free runs in order to guarantee that agreement will be achieved, even when there are (up to t) faulty processes.

4. Lower bounds

The main result of this section is the following theorem:

Theorem 1. *If $t < n-1$, the sum of the number of messages in the failure-free runs of any t-resilent protocol for binary BA under crash failures is at least $n+t-1$.*

[1] If $v \in \{0, 1\}$ then $\overline{v} \stackrel{\text{def}}{=} 1 - v$ denotes the complement of v.

We first establish some useful notation and terminology. Fix an arbitrary t-resilient protocol P for binary BA under crash failures. Everything we define and prove in this section is relative to this protocol, but as P is arbitrary the results apply in general.

In our proof we shall count the number of messages used in the failure-free runs H_0 and H_1 by dividing the processes (possibly excluding the sender) into three groups according to the number of messages they receive in these runs. More specifically, we define three disjoint sets of processes, R_0, R_1 and B, as follows:

- R_v is the set of receivers that receive no message in H_v and receive exactly one message in $H_{\overline{v}}$, for $v \in \{0, 1\}$.
- B is the set of receivers that are not in R_0 or R_1 together with the sender, if the sender receives a message in at least one of the failure-free runs.

Since, as we noted in Section 3, every receiver must receive a message in one of the two failure-free runs, B consists of the receivers which receive at least two messages in H_0 and H_1 combined and, possibly, the sender. Thus, R_0, R_1 and B form a partition of the set of all processes or of all receivers, according as the sender is a member of B or not.

The main lemma we shall use in the proof of Theorem 1 is Lemma 2 below. Essentially it states that the only processes that can directly detect that a run is not failure-free are the members of B. The other processes can only find out that there has been a failure by being told about it by some process that had previously learned this. More precisely, if we consider a run in which there are no failures up to a certain time, and in which we prevent all members of B from sending anything after this time (by crashing them, if necessary) then, as long as no more than t failures have occurred, none of the correct processes in R_0 or R_1 will be able to tell that they are not in a failure-free run.

We say that a process p *detects a failure in round l of run E* if

a. $\mathbf{view}(p, E^{l-1}) = \mathbf{view}(p, H_v^{l-1})$ for some $v \in \{0, 1\}$,

b. $\mathbf{view}(p, E^{l}) \neq \mathbf{view}(p, H_v^{l})$ for both $v \in \{0, 1\}$, and

c. p hasn't already crashed before round l of E.

We say that p *detects a failure in E* if it does so in some round l. Note that, by this definition, a faulty process is said to detect a failure in E only if it had done so before it crashed. This turns out to be a convenient convention.

Lemma 2. *Suppose E is a run of the protocol, $v \in \{0, 1\}$, and i is a round such that*

1. *E and H_v agree through round $i - 1$, and*
2. *no process in B sends a message after round i.*

Then no $p \in R_0 \cup R_1$ detects a failure in E.

Note that if $|B| \leq t$ then a run satisfying the conditions of the Lemma exists: We can construct such a run by ensuring that the processes in B do not send any messages after round i simply by crashing them, if necessary. This will be a valid execution because the condition $|B| \leq t$ means that all the processes in B can crash without exceeding the limit t on the number of faulty processes.

Proof: Assume the lemma is false and let $v \in \{0, 1\}$, i be a round number, and E a run that satisfies the conditions of the Lemma, but in which there exists a correct process $p \in R_0 \cup R_1$ which detects a failure in some round l of E. Without loss of generality, we can assume that no process detects a failure in E before round l. Note that $l \geq i$, because by assumption (1) of the Lemma runs E and H_v agree through round $i-1$ so no process can detect a failure in E before round i.

Claim. *A process that sends messages after round i in E cannot detect a failure in E before p does.*

Proof of the Claim: By the definition of E only processes in $R_0 \cup R_1$ and the sender (if it is not in B) may send messages after round i in E. By the choice of p, no process in $R_0 \cup R_1$ can detect a failure in E before p does. Thus, it remains to show that if the sender is not in B it does not detect a failure in E before p does.

If the sender never detects a failure in E, we are done. If it does, let l' be the round when that occurs. Since the sender is not in B, i.e., it does not receive any message in either H_0 or H_1, it must receive a message in E — say the message is received from process q in round l'.

q cannot be in B. To see this, first note that $l' \geq i$ (because E and H_v are identical through round $i-1$). If $l' > i$ then by assumption (2) of the Lemma, q cannot be in B. If $l' = i$ then q would send a message to the sender on the basis of $\mathbf{view}(q, E^{i-1})$ which is equal to $\mathbf{view}(q, H_v^{i-1})$. Thus, q would send a message to the sender in round i of H_v, which would mean that the sender is in B, a contradiction.

Also, q cannot be the sender itself because if it was, it would be sending a message to itself before having detected a failure, which means that the sender would receive a message (from itself) in some failure-free run and would therefore be in B, a contradiction.

Hence, q must be in $R_0 \cup R_1$. Since the sender does not receive any message in either H_0 or H_1, the view of q in $E^{l'-1}$ must be different from that in $H_0^{l'-1}$ or $H_1^{l'-1}$ and so, q detects a failure in E no later than round $l'-1$.

By choice of l as the earliest round in E in which some process detects a failure, $l' - 1 \geq l$. Therefore, $l' > l$, as wanted. ▮ Claim

Next we argue that p must receive exactly one message by the end of round l in E. Since $p \in R_0 \cup R_1$, p receives no message in some failure-free run. If p received no message by the end of round l in E it would not have detected a failure in round l of E, contrary to the choice of p and l. Therefore, p receives at least one message by the end of round l in E.

On the other hand, p cannot have received more than one message by that round. For, suppose it did. Let one such message be sent by process q in round $r \leq l$ and another be sent by q' in round $r' \leq l$. By the Claim, $\mathbf{view}(q, E^{r-1}) = \mathbf{view}(q, H_u^{r-1})$ and $\mathbf{view}(q', E^{r'-1}) = \mathbf{view}(q', H_{u'}^{r'-1})$, for some $u, u' \in \{0, 1\}$. But then p receives at least two messages in H_0 and H_1 combined, which would make it a member of B, not of $R_0 \cup R_1$. Thus, p can't receive more than one message by the end of round l in E. Since, as we saw in the preceding paragraph, it must receive at least one message by that round, it must actually receive exactly one — say one sent by process q in round $r \leq l$.

By the Claim, $\mathbf{view}(q, E^{r-1}) = \mathbf{view}(q, H_u^{r-1})$, for some $u \in \{0, 1\}$. Therefore, q sends the same message to p in round r of E as in round r of H_u. Since $p \in R_0 \cup R_1$ and p receives a message in H_u, in fact, $p \in R_{\overline{u}}$. By definition of $R_{\overline{u}}$, p receives exactly one message in H_u, the one sent by q in round r. But this is exactly what happens in E through round l. Thus, $\mathbf{view}(p, E^l) = \mathbf{view}(p, H_u^l)$, which means that p does not detect a failure in round l of E, contrary to the choice of p and l. ∎

Lemma 2 is the main tool by which we shall prove Theorem 1. We shall construct runs in which everything occurs correctly until some processes have decided upon their values, and then "silence" the processes in B, in an attempt to force the undecided processes to reach conflicting decisions, thereby violating agreement.

Proof of Theorem 1: Let $s = |R_0| + |R_1| + |B| - n + 1$; that is, s is 1 or 0 according as the sender is in B or not. From the definitions of R_0, R_1 and B, the total number of messages in the two failure-free runs combined is at least

$$|R_0| + |R_1| + 2|B| - s = (n + s - 1 + |B|) - s = n + |B| - 1. \tag{1}$$

The Theorem follows immediately if $|B| \geq t$. Let us then assume the alternative, namely that $|B| \leq t - 1$. We shall show that this assumption leads to a contradiction.

From the definition of s we have

$$|R_0| + |R_1| = n + s - 1 - |B| \geq n - t + s \geq 2 + s. \tag{2}$$

Each process in $R_0 \cup R_1$ expects a message in exactly one failure-free run. Let p be a process in $R_0 \cup R_1$ which expects that message at least as late as any other. More precisely, let p be a process and $v \in \{0, 1\}$ be so that:

a. $p \in R_{\overline{v}}$,

b. no process in $R_{\overline{v}}$ receives its message in H_v in a round later than than in which p receives its message in H_v, and

c. no process in R_v receives its message in $H_{\overline{v}}$ in a round later than than in which p receives its message in H_v.

By (2) at least one of R_0 or R_1 is non-empty and therefore a process as defined above must exist. (There may be more than one process and initial value which satisfy (a)–(c); in this case we choose p and v arbitrarily among all those processes and values that do.) Let q be the process that sends the message to p in H_v, and i be the round in which that message is sent.

We shall now identify another run E of the protocol, as follows:

- E agrees with H_v through round $i - 1$,
- no process in B sends a message after round i,
- q crashes in round i having sent all the messages it sends in round i of H_v (if any) *except* the message it sends to p,

- only the minimum number of crashes necessary to satisfy the preceding conditions occur.

The only processes that may be faulty in E are the members of B and q. Since we assume that $|B| \leq t-1$, no more that t processes are faulty in E, so E is a legitimate run of the protocol.

Run E satisfies the conditions of Lemma 2, so we apply it to conclude that every member of $R_0 \cup R_1$ which is correct in E has a view identical to that in one of the failure-free runs. This is the case, in particular, for p. Since p receives a message from q in round i of H_v, while it receives no such message in E, it must be that $\mathbf{view}(p, E) = \mathbf{view}(p, H_{\overline{v}})$. Therefore,

$$p \text{ decides } \overline{v} \text{ in } E. \tag{3}$$

If the sender is correct in E then Validity is violated. For, since E and H_v agree in the first $i-1$ rounds, the sender has the same initial value in both, i.e., v. Thus, every correct process in E should decide v, contradicting (3).

We may therefore assume that the sender is not correct in E. Now we shall show that there must be some process, other than p, which is in $R_0 \cup R_1$ and is correct in E. To see this, consider two cases:

Case 1: The sender is not in B. Since, by assumption, the sender is faulty and the only process not in B which may be faulty is q, it follows that q is precisely the sender. Therefore, every process in $R_0 \cup R_1$ is correct. By (2), $R_0 \cup R_1$ contains at least two processes, both of which are correct.

Case 2: The sender is in B. Then $s = 1$ and by (2), $|R_0 \cup R_1| \geq 3$. The only process in $R_0 \cup R_1$ which may be faulty is q, so $R_0 \cup R_1$ contains at least two correct processes.

Let p' be a process, other than p, which is in $R_0 \cup R_1$ and is correct in E. We claim that p' decides v in E. To see this we again consider two cases:

Case 1: $p' \in R_{\overline{v}}$. Process p' receives a message in E (because the only process in $R_{\overline{v}}$ which does not receive in E the message which it receives in H_v is p) while it receives no message in $H_{\overline{v}}$. By Lemma 2, p' has the same view in E as in one of the two failure-free runs. Since it can't have the same view as in $H_{\overline{v}}$ it must have the same view as in H_v. This means that p' decides v in E.

Case 2: $p' \in R_v$. By Lemma 2, p' has the same view in E as in one of the failure-free runs. It is not possible for p' to receive a message in E, for such a message would have to be received after round i (since up through round i, any message sent it E is also sent in H_v and p', being a member of R_v, receives no message in H_v) and p was chosen so that no process in $R_0 \cup R_1$ receives a message in a failure-free run after round i. Therefore, p' has the same view in E as in H_v (in which it also receives no message) and therefore decides v in E.

We have shown that p' decides v in E. In conjunction with (3), this contradicts Agreement (recall that p and p' are both correct in E). We arrived at this contradiction by assuming that $|B| \leq t-1$. Therefore, $|B| \geq t$ and the theorem follows by (1). ∎

Theorem 1 gives a lower bound on the total message complexity but it also immediately implies lower bounds on the worst case and average case message complexities.

Corollary 3. *If $t < n-1$, the worst case failure-free message complexity of any t-resilent protocol for binary BA under crash failures is at least* $\lceil (n+t-1)/2 \rceil$. ∎

Corollary 4. *Let P_0 and P_1 be the probability that the sender's initial value in a failure-free run is 0 or 1, respectively. If $t < n-1$, the average case message complexity of any t-resilient protocol for binary BA under crash failures is at least* $\min(P_0, P_1) \cdot (n+t-1)$.

Proof: Let M_v be the number of messages in H_v, where $v \in \{0,1\}$. By Theorem 1, $M_0 + M_1 \geq (n+t-1)$. The expected number of messages in failure-free runs is:

$$P_0 M_0 + P_1 M_1 \geq \min(P_0, P_1) \cdot (M_0 + M_1) \geq \min(P_0, P_1) \cdot (n+t-1). \qquad \blacksquare$$

5. Upper Bounds

In this section we present two t-resilient binary BA protocols for crash failures. The first protocol uses at most $\lceil (n+t-1)/2 \rceil$ messages in each of the failure-free runs. In light of Corollary 3, it is optimal as far as worst case message complexity is concerned. The second protocol uses an expected number of $\min(P_0, P_1) \cdot (n+t-1)$ messages in the failure-free runs, so in light of Corollary 4, it is optimal from the point of view of average case message complexity.

5.1 Worst case optimal protocol

In this protocol, the receivers are divided into three disjoint groups B, R_0, and R_1. (These names are chosen to emphasise the correspondence between this protocol and the proof of the lower bound.) B contains exactly t receivers while R_0 and R_1 each contain half of the remaining receivers — with one of the two, say R_0, receiving an extra one, if $n-t-1$ is odd.

As in the protocol presented in Section 3 for the case where there are no failures at all, a process in R_v expects to hear from the sender in round 1 only if the initial value is $\overline{v}$; in the absence of any message such a process will decide v. By contrast, a process in B expects to hear from the sender no matter what the initial value is. The sender sends its initial value v to $R_{\overline{v}}$ in the *first* round and to B in the *second* round. This ordering is important as will be seen from the following informal argument for the protocol's correctness. There are three possibilities:

a. All processes in B are faulty. Since $|B| = t$ this means that the sender must be correct and will therefore send its initial value v to all processes in $R_{\overline{v}}$ but not to any process in R_v. Thus, all correct processes will decide v.

b. All correct processes in B receive the sender's initial value v in round 2. Since we admit only crash failures, the sender, even if faulty, could not have crashed in round 1.

Then it must have sent its initial value to all processes in $R_{\overline{v}}$ and to no process in R_v. Again all correct processes will decide v.

c. Some correct process p in B does not receive a message from the sender in round 2. But then such a process has determined that the sender is faulty and that the current run is not failure-free. Thus, p can "take over" as the sender and invoke any standard BA protocol that works for crash failures (cf. [14] for such protocols).

Since our stated goal in this paper is to minimise the number of messages in the failure-free runs, we don't really care how many messages are used by the protocol invoked in case (c), as that case involves only non-failure-free runs. Nevertheless, we shall present a protocol based on this idea which is economical in the number of messages even when there are failures. The key is to prevent many processes in B, all of which detect that the sender is faulty in round 2, from initiating BA protocols simultaneously, thus generating a flurry of messages. This is achieved by ensuring that processes in B "take over" the sender's role in an orderly fashion.

To be more concrete, let the t processes in B be numbered $p_1, p_2, \ldots, p_t$. Each process p_j in B waits until round $4j-2$ for a message telling it what to decide. If no message arrives during this time then p_j knows that the sender is faulty. It then tries to find out whether any other member of B has heard the sender's initial value. It does this by sending the message "request" to all members of B in round $4j-1$. Any correct member of B which receives such a request and has heard the sender's value responds by sending back to p_j a message containing that value in round $4j$. If p_j does not receive any reply, then it knows that the sender is faulty and that no process in B which has not already crashed knows the sender's initial value. Therefore, it is safe for p_j to just arbitrarily choose a value from $\{0,1\}$ and force it upon all processes. We shall denote this value as **default**. Thus, in this case p_j sends **default** to all members of R_0 and R_1 in round $4j+1$ and to all members of B in round $4j+2$.

Once p_j has determined the decision value, whether by receiving a message directly from the sender, getting a response to a "request" message, or using **default**, it then spends the remaining rounds until round $4t+2$ by responding to any "request" messages that it might receive.

In this section we shall refer to this as protocol WC (for "worst case"); it is given in pseudo-code in Figure 1. In the figure, *init* is a variable which is local to the sender and contains its initial value. Each receiver has a local variable called *state* which, at the end of the protocol, will contain the value which the receiver will decide; this is simply assigned to the *decision* variable of that receiver.

Next we show that protocol WC is correct and uses no more than $\lceil (n+t-1)/2 \rceil$ messages in any failure-free run. First we need a definition and some preliminary results.

We say that a process p in B *knows* the sender's value v if either p receives from the sender the message v in round 2, or it receives a message v from some other member in B which (recursively) knows the sender's value.

Lemma 5. *Every correct receiver has* $state = v$ *or* $\overline{v}$ *at the end of the protocol.*

Proof: This is obvious for processes in R_0 and R_1 since these are the only possible values

Algorithm for the sender:

in round 1: **send** $init$ to all processes in $R_{\overline{init}}$
in round 2: **send** $init$ to all processes in B
at the end of round 2 set $decision := init$ and **halt**.

Algorithm for members of R_v, $v \in \{0, 1\}$:

($state$ is initialised to v before round 1.)
in round $1 \leq r \leq 4t + 2$: **if** message u is received **then**
  $state := u$
at the end of round $4t + 2$ set $decision := state$ and **halt**.

Algorithm for process p_j in B, $1 \leq j \leq t$:

($state$ is initialised to **undefined** before round 1.)
in round $1 \leq r < 4j - 1$: **if** message v is received **then**
  $state := v$
  elsif message "request" is received from process q **and** $state \neq$ **undefined then**
  send reply $state$ to q
in round $4j - 1$: **if** $state =$ **undefined then**
  send message "request" to all processes in B
in round $4j$: **if** a reply v to a "request" message is received **then**
  $state := v$
in round $4j + 1$: **if** $state =$ **undefined then**
  send default to all members of R_0 and R_1
in round $4j + 2$: **if** $state =$ **undefined then**
  send default to all members of B
  $state :=$ **default**
in round $4j + 2 < r \leq 4t + 2$: **if** message "request" is received from some process q **then**
  send reply $state$ to q
at the end of round $4t + 2$ set $decision := state$ and **halt**.

Figure 1: Protocol WC

of $state$. A process in B might have $state =$ **undefined**. However, in round $4i + 2$, p_i will set its state to **default**, if it has not crashed or already set it to 0 or 1. Further, once set to 0 or 1, $state$ cannot be reset to **undefined**. ∎

Lemma 6. *If any correct process in B knows the sender's value v then all correct receivers will decide v.*

Proof: Let p_i be a correct process in B which knows the sender's value, say v. This means that the sender must have sent its value to some process in B in round 2. Since we only admit crash failures, the sender must have sent v to all processes in $R_{\overline{v}}$ in round 1. Thus all correct processes in $R_{\overline{v}}$ will set their $state$ to v at the end of round 1. Further, no process can set its state to $\overline{v}$. For, given that the sender's initial value was v, this would happen only if some p_j sent **default** in round $4j + 1$ (with **default** $= \overline{v}$). However, before doing so, p_j would have sent "request" messages to all processes in B and if any of them knew the sender's value, it would have responded and p_j would not have sent **default**. Therefore, any process that knows the sender's value would have crashed before round $4j+1$, contradicting the existence of p_i, a process which knows the sender's value but never crashes. Since no process can set its value to $\overline{v}$ and, as we have seen, all correct processes in $R_{\overline{v}}$ — which are the only ones to initialise their state to $\overline{v}$ — will set their state to v in round 2, it follows that no correct receiver can have *state* $= \overline{v}$ at the end of the algorithm. By Lemma 5 then, all correct receivers will decide v. ∎

Theorem 7. *Protocol WC is a correct t-resilient binary BA protocol under crash failures.*

Proof: Termination holds, by Lemma 5.

Regarding Validity, if the sender is not faulty and has initial value v then it sends v to all members of $R_{\overline{v}}$ and to all members of B. Since every member of B will receive a message from the sender, none of them will send any additional messages. Therefore every correct process in R_v will not receive any messages, and so will decide v; while every correct process in $R_{\overline{v}}$ or B will receive the sender's message, and decide v.

Agreement follows by Validity, in case the sender is correct. If the sender is faulty, since $|B| = t$ and at most t processes can fail, at least one process in B is correct. If a correct process in B knows the sender's value then Agreement follows by Lemma 6. Otherwise, there is a process $p_i \in B$ which sends **default** to all processes in B in round $4i+2$ and thus to all processes in R_0 and R_1 in round $4i+1$. So at the end of round $4i+2$, all correct processes will have their state set to **default** and will all agree on that value. ∎

Theorem 8. *Protocol WC uses no more than $\lceil (n+t-1)/2 \rceil$ messages in any failure-free run.*

Proof: Say the sender's initial value is v, and no process fails. Then the sender sends one message to each process in $R_{\overline{v}}$, and one message to each process in B. None of the processes in R_0 or R_1 send any messages, and since every process in B will receive the sender's message, none of these processes will send messages either. Thus, the only messages sent are those sent by the sender. There are $|R_{\overline{v}}| + |B|$ such messages. $|R_{\overline{v}}| \leq \lceil (n-t-1)/2 \rceil$, and $|B| = t$, so no more than $t + \lceil (n-t-1)/2 \rceil = \lceil (n+t-1)/2 \rceil$ messages are sent in any failure-free run. ∎

Now we consider the bit complexity of this protocol. Messages need to convey the sender's initial value (0 or 1), the state of a process in B, and the message "request". First of all, even though there are three possible states for processes in B, we note that a process $p_j \in B$ never sends its state while that is still **undefined**. Thus, the only values that actually may be contained in messages are 0, 1 and "request". It would seem that at least two bits are needed to distinguish these three values. Notice, however, that the message "request" can only be sent in a round r only if $r \equiv 3 \bmod 4$. Furthermore, no other type of message may be sent in such a round. Therefore, we can use either 0 or 1, when sent in round r with $r \equiv 3 \bmod 4$, to signify a "request" message. In conclusion, each message needs to carry a single bit. Therefore, the worst case *bit* complexity of protocol *WC* is at most $\lceil (n+t-1)/2 \rceil$. In view of Corollary 4, and the obvious fact that a lower bound on (any type) of message complexity is a lower bound on bit complexity, it follows that WC is optimal for worst case bit complexity. It is also easy to see that it is optimal for total complexity (both message and bit).

We conclude our discussion of WC with some comments on its message complexity in runs that contain failures. More specifically, we want to determine the maximum number of messages generated by this protocol when f processes are faulty, for any $1 \leq f \leq t$. Consider the four rounds during which p_j is "in charge", namely rounds $4j-1$ through $4j+2$. No messages are sent during these rounds if p_j has its *state* set to 0 or 1. Thus, to maximise the number of messages sent, we should keep as many processes in B with *state* = **undefined** for as long as possible. By this reasoning, the run with the worst case number of messages when f processes fail has the following characteristics:

- The sender crashes at the beginning of round 2 (leaving all processes in B with *state* = **undefined**).
- For $1 \leq j < f$, process p_j sends t "request" messages in round $4j-1$, receives no replies in round $4j$ (since all processes in B have *state* = **undefined**), sends $n-t-1$ defaults to $R_0 \cup R_1$ in round $4j+1$, and then crashes sending no messages in round $4j+2$ (thereby leaving again all processes in B with *state* = **undefined**).

The next process in B, p_f, will behave in the same way except that, since we have exhausted the number f of processes that can be faulty, p_f will succeed in sending its default value to all processes in B in round $4f+2$. Since all correct processes in B will then have *state* $\neq$ **undefined**, no messages are subsequently generated. Therefore, this scenario involves in a total of

$$\lceil (n-t-1)/2 \rceil + (f-1)\big(t + (n-t-1)\big) + (t+n-1)$$

messages. (The first term represents the number of messages sent by the sender in round 1, the second term represents the number of messages sent by $p_1, \ldots, p_{f-1}$, and the third term represents the number of messages sent by p_f.) Straightforward algebraic manipulations simplify this to $\lceil (n+t-1)/2 \rceil + f(n-1)$. Thus, in WC each process failure costs a maximum of $n-1$ messages in addition to those that must be sent in the failure-free runs.

5.2 Average case optimal protocol

Next we present a protocol that is optimal from the point of view of average case message complexity. Let P_0 and P_1 be the probabilities that the sender's initial value is 0 and 1, respectively. We assume that these are fixed *a priori* and are independent of failures. Thus, the probability of a failure-free run being H_v is P_v, for $v \in \{0, 1\}$.

Let v be the initial value that has greater probability (v can be either value if $P_0 = P_1$). To minimise the expected number of messages in the failure-free runs, we would like H_v to have as few messages as possible, and $H_{\overline{v}}$ to have the remaining messages (to make up the $n+t-1$ lower bound on the total complexity). It is actually possible to construct a protocol which uses no messages at all in H_v and uses $n+t-1$ messages in $H_{\overline{v}}$. This, of course, is the best we can hope for.

We group the receivers into two sets: B, containing t receivers, and R_v, containing the remaining $n-t-1$ receivers. If the initial value is v, the sender sends no messages

— it simply decides v. If the initial value is $\overline{v}$, the sender sends $\overline{v}$ to all processes in B in round 1; then it sends $\overline{v}$ to all processes in R_v in round 2; finally, it sends again $\overline{v}$ to all processes in B in round 3 and decides $\overline{v}$. A process in R_v decides v if it receives no message and $\overline{v}$ otherwise. A process in B decides v if it receives no message and $\overline{v}$ if it receives messages in both rounds 1 and 3. If it receives $\overline{v}$ only in round 1, it knows that the sender must be faulty.

Let us see informally why this works. There are four cases.

a. All processes in B are faulty. Since $|B| = t$, this means that all other processes must be correct. In particular, if the initial value is v the sender will send no message at all and thus all processes in R_v will decide v; if the initial value is $\overline{v}$, the sender will send this value to all processes in R_v and again these will all decide $\overline{v}$. Thus in this case all correct processes will agree.

b. Some processes in B are correct but none of them receives any message from the sender. Since only crash failures can occur and messages can be sent to processes in R_v only after they are sent to the processes in B, this means that no process in R_v received a message either. Thus, all correct processes will agree on v.

c. Some processes in B are correct and all receive a message from the sender in both rounds 1 and 3. Again, since the messages to R_v are sandwiched in a round between the messages to B and there are only crash failures, the fact that all correct processes in B received a message in both rounds 1 and 3 means that all correct processes in R_v also received a message in round 2. Now all correct processes will agree on $\overline{v}$.

d. Some correct process in B receives a message in round 1 but not in round 3. Such a process knows that the sender must have crashed somewhere between rounds 1 and 3. In any case, it knows that this is not a failure-free run and can now invoke any standard BA protocol that works for crash failures to force agreement.

Even though in case (d) we can use any BA whatsoever without affecting our goal of minimising the number of messages in failure-free runs, we shall give a protocol extending this idea which uses few messages even in non-failure-free runs. As in the previous subsection, the idea is to coordinate the processes in B so that they don't all start the standard BA protocol upon detecting a failure in the same round. More concretely, let us number the t processes in B as $p_1, p_2, \ldots, p_t$. Process p_j can be in one of three states: v, **worried** or $\overline{v}$. Informally, the meaning of these states is:

- p_j will be in state v if it is certain that no process in R_v has received $\overline{v}$. This will be the case if p_j itself has not received such a message or if it receives a message from another process in B telling it that it is safe to be in state v (because that other process is certain that no process in R_v has received $\overline{v}$).
- p_j will be **worried** if it considers it possible that some processes in R_v have received $\overline{v}$ while others have not. This will be the case, for example, if p_j gets $\overline{v}$ from the sender in round 1 but does not receive that message in round 3. This represents a potentially dangerous situation since — unless rectified — it would result in inconsistent decisions by the processes in R_v. In general, as we shall see shortly, p_j will be **worried** if it received $\overline{v}$ only in round 1.

- Finally, p_j will be in state $\overline{v}$ if it is certain that all processes in R_v have received $\overline{v}$. This will be the case if it received $\overline{v}$ in round 1 *and* in some later round.

To achieve this effect, each $p_j \in B$ will wait for its "turn", consisting of four successive rounds: $4j$ through $4j+3$. If a process is not **worried** when its turn comes, it does nothing. If, on the other hand, it is **worried**, it will first attempt to ascertain if there is any other process in B which has not crashed and is not **worried**; it does this by sending "request" messages in round $4j$. Any process in B which receives such a request and is not **worried**, responds by sending its state in round $4j+1$. If p_j receives any reply, it simply adopts the state reported there. (It turns out that it is impossible for it to receive conflicting states, so there is no ambiguity here.) If, on the other hand, p_j receives no reply, it knows that all processes in B which have not crashed yet are **worried**. It will then try to salvage the situation by completing what the sender may have failed to do in rounds 2 and 3. Namely, it will broadcast $\overline{v}$ to the processes in R_v in round $4j+2$ and then to the processes in B in round $4j+3$.

At the end of the protocol, i.e., in round $4t+2$, each process decides according to its *state*.

The protocol, called AC_v (for "average case", with the subscript indicating the more likely of the two decision values), is given in pseudo-code in Figure 2.

To prove that protocol AC_v is correct we need some preliminary results.

Lemma 9. *Every correct receiver will be in state v or $\overline{v}$ by the end of the protocol.*

Proof: For processes in R_v, this is obvious because these are the only possible values of *state*. For a process in B, *state* may also have the value **worried**. However, it is easy to see that after its "turn", i.e., at the end of round $4i+3$, p_i will not be **worried** (unless it has crashed), because if it is **worried** in round $4i$ it will set its value to v or $\overline{v}$ in round $4i+1$ or $4i+3$. Furthermore, *state* cannot be reset to **worried**. So, by round $4t+3$, all correct processes in B will be in state v or $\overline{v}$. ∎

Lemma 10. *For any $0 \leq j \leq t$, if p_j sends $\overline{v}$ in round $4j+3$ then every correct receiver will decide $\overline{v}$, where we take p_0 to be the sender.*

Proof: If p_j sends $\overline{v}$ in round $4j+3$ then, since only crash failures can occur, it must have sent $\overline{v}$ to all processes in R_v in the previous round, $4j+2$. Thus all correct processes in R_v are in state $\overline{v}$ at the end of that round. Since processes in R_v can not change their state once it is set to $\overline{v}$, all correct processes in R_v will decide $\overline{v}$. It remains to show that the same holds for correct processes in B.

Let B_r denote the set of processes in B which have not crashed by the end of round r. That is, these are the processes which may still send messages after round r. First, we shall show that all processes in B_{4j+1} are *not* in state v at the end of round $4j+1$. There are two cases:

Case 1: $j=0$. Then the hypothesis of the Lemma says that the sender sends $\overline{v}$ in round 3. Since we only admit crash failures, the sender must have sent $\overline{v}$ to all processes in B in round 1. Thus, all processes in B_1 will be in state **worried**, and not in v, at the end of round 1.

Algorithm for the sender:

if $init = \overline{v}$ **then**

in round 1: **send** $\overline{v}$ to all processes in B

in round 2: **send** $\overline{v}$ to all processes in R_v

in round 3: **send** $\overline{v}$ to all processes in B

$decision := \overline{v}$

else

$decision := v$

halt

Algorithm for members of R_v:

($state$ is initialised to v before round 1.)

in round $1 \leq r \leq 4t+3$: **if** message $\overline{v}$ is received **then**

$state := \overline{v}$

at the end of round $4t+3$ set $decision := state$ and **halt**.

Algorithm for process p_j in B, $1 \leq j \leq t$:

($state$ is initialised to v before round 1.)

in round 1: **if** message $\overline{v}$ is received **then**

$state :=$ **worried**

in round $1 < r < 4j$: **if** message $\overline{v}$ is received **then**

$state := \overline{v}$

elsif message "request" is received from process q **and** $state \neq$ **worried then**

send reply $state$ to q

in round $4j$: **if** $state =$ **worried then**

send "request" message to all processes in B

in round $4j+1$: **if** a reply u to a "request" message is received **then**

$state := u$†

in round $4j+2$: **if** $state =$ **worried then**

send message $\overline{v}$ to all processes in R_v

in round $4j+3$: **if** $state =$ **worried then**

send message $\overline{v}$ to all processes in B

$state := \overline{v}$

in round $4j+3 < r \leq 4t+3$: **if** message "request" is received from process q **then**

send reply $state$ to q

at the end of round $4t+3$ set $decision := state$ and **halt**.

† We cannot *a priori* exclude the possibility that two conflicting replies will be received. It turns out that this is impossible but we shall not explicitly prove this fact, as it is not necessary for our purposes. To make this step well-defined, this should be understood as: pick an arbitrary reply among those received and assign the value contained in it to variable *state*.

Figure 2: Protocol AC_v

Case 2: $j > 0$. Then the hypothesis of the Lemma says that $p_j \in B$ sends $\overline{v}$ in round $4j+3$. Since we only admit crash failures, p_j must have sent a "request" message in round $4j$ to all processes in B none of which replied in round $4j+1$. Thus, no process in B_{4j+1} could be in state v in round $4j+1$ (otherwise, it would have replied).

Next we show that no process in B_{4j+1} can be in state v after round $4j+1$. For, suppose there was such a process. Since, as we just showed, all processes in B_{4j+1} are not in state v at the end of round $4j+1$, there must be some process in B_{4j+1} which first changes its state to v after that round. Let p_k be that process. The only way this can happen is if p_k receives a reply containing v from some process $p_l \in B$ in round $4k+1 > 4j+1$. But since p_l sent a message in round $4k+1$ it must be in B_{4j+1} (by the nature of crash failures) and it must therefore have been in state v before that round, contradicting the choice of p_k as the first process in B_{4j+1} to have changed its state to v.

Thus, no process in B_{4j+1} can be in state v at the end of the protocol. In particular, no correct process in B can be in that state. Lemma 9 then implies that all correct processes in B will be in state $\overline{v}$ at the end of the protocol and will therefore decide $\overline{v}$.

Theorem 11. *Protocol AC_v is a t-resilient binary BA protocol for crash failures.*

Proof: Termination holds by Lemma 9.

For Validity, suppose that the sender is correct. If the initial value is v then no message will ever be sent and all processes will decide v. If the initial value is $\overline{v}$, then the sender will send $\overline{v}$ in round 3. By Lemma 10 (with $j=0$) all correct processes will decide $\overline{v}$.

Agreement follows from Validity in case the sender is correct. If the sender is faulty we only care about agreement among correct receivers. If any correct process decides $\overline{v}$ then p_j must have sent $\overline{v}$ in round $4j+3$, for some $0 \le j \le t$ (where we take p_0 to be the sender). By Lemma 10, all correct receivers will decide $\overline{v}$. Having established agreement for one value, agreement for the other follows immediately (given Termination). ∎

Regarding the number of messages in the failure-free runs of AC_v we have:

Theorem 12. *Protocol AC_v requires an expected number of $P_{\overline{v}} \cdot (n+t-1)$ messages in the failure-free runs.*

Proof: It is obvious that if the initial value is v then no messages are sent at all. If the initial value is $\overline{v}$ and the sender is correct, by the end of round 3 all receivers will be in state $\overline{v}$, so no messages will be sent after that round by any process. The sender sends $\overline{v}$ to each process in R_v once and to each process in B twice. So the total number of messages in this failure free run is $(n-t-1)+2t = n+t-1$. Thus, the expected number of messages in the failure-free runs is $P_v \cdot 0 + P_{\overline{v}} \cdot (n+t-1) = P_{\overline{v}} \cdot (n+t-1)$. ∎

In light of Corollary 4, this is the best possible from the point of view of average message complexity. Regarding the bit complexity of this protocol, we can implement it with just one bit in each message, by using the round number to encode the fact that a "request" message (as opposed to a "value" message) is being sent. In particular, a message is a "request" one if and only if it is sent in a round $r \neq 1$ such that $r \equiv 1 \bmod 4$. Thus, AC_v is also optimal for average bit complexity. It is easy to see that, like WC, it is optimal for total message and bit complexity as well. An expression for the maximum number of messages sent when there are f failures, $1 \leq f \leq t$, can be derived as for WC. The details are left to the interested reader.

6. Open problems

One direction of extending our results is to consider non-binary BA. Let V be the set of possible decision values (in the case of binary BA, $V = \{0, 1\}$). It is easy to see that if $|V| > n-1$ then $n-1$ is an exact bound on the worst case number of messages in failure-free runs. To prove that this is a lower bound, it is enough to show that, in any protocol, there is some failure-free run in which every process receives a message. First, notice that there can be at most one failure-free run in which a receiver p receives no message: otherwise there would be two distinct failure-free runs in both of which p would have the same view (i.e., no messages received in any round) and would therefore have to decide the same value, violating Validity in at least one of them. Since there are more than $n-1$ possible decision values, there must be at least one failure-free run in which each of the $n-1$ receivers actually receives a message. It is easy to design a protocol that solves the problem using no more than $n-1$ messages in the worst case in any failure-free run. For example, we can use protocol P of Section 5.1, where all the receivers are in B and $R_0 = R_1 = \emptyset$. Therefore the lower bound of $n-1$ messages in the worst case is tight, if $|V| > n-1$.

If $2 < |V| \leq n-1$ then we can modify the protocol of Section 5.1 as follows. The receivers other than the t that belong to B are divided up into $|V|$ sets, each corresponding to some $v \in V$ and containing either $\lceil (n-t-1)/|V| \rceil$ or $\lfloor (n-t-1)/|V| \rfloor$ processes. Let R_v be the set of processes that correspond to decision value v. As before, the receivers in R_v will decide v unless they receive a message stating otherwise. When the sender's initial value is v, it sends v to all processes except those in R_v, which decide v by default. Thus we obtain an upper bound of

$$n - 1 - \lfloor (n-t-1)/|V| \rfloor = \lceil \big((|V|-1)(n-1) + t \big) / |V| \rceil$$

on the worst case number of messages in this case. We do not know if this is best possible, however. Our proof technique for the lower bound in the binary case does not generalise.

Another direction for research is to consider the failure-free message complexity of BA under failure modes more general than crash failures. Of particular interest is the generalisation of crash failures known as *sending omission failures*. In this case, a faulty process may fail to send some messages in one or more rounds. However, unlike crash failures, failing to send a message in one round does not mean that the process will not send any messages again. In some sense, failures are "intermittent". Since crash failures are a special case of sending omission failures, the lower bounds in this paper are also lower bounds for sending omission failures. However, the correctness of our protocols depends very crucially on the fact that a faulty process, once it fails to send a message in one round, sends no messages in subsequent rounds. It would be useful to derive matching bounds for the various measures of failure-free message complexity of BA protocols that work for this class of failures. The same question can be asked about other, more general, types of failures.[2]

Yet another direction for research concerns bounds on the number of messages *over the entire set of runs of a protocol* — not just the failure-free ones. For the worst case message complexity over all runs, Dolev and Reischuk [6] proved a tight asymptotic bound of $\Theta(n+t^2)$ messages if the processes are subject to *general omission failures*, a generalisation of sending omission failures. More specifically, in this case, a process may fail to send a message its protocol stipulates should be sent; or it may fail to *receive* a message that was sent to it in that round.[3] For crash failures, there is an obvious lower bound of $\Omega(n)$ messages. There is no hope of extending the $\Omega(n + t^2)$ bound of Dolev and Reischuk to this case, since there are algorithms that tolerate crash failures and use only $O(n + t\sqrt{t})$ messages in the worst case over all runs [3]. We do not know if this is a tight asymptotic bound for crash failures.

The gap between upper and lower bounds on the worst case message complexity of BA over all runs is even greater in the case of sending omission failures. Here, the best known lower bound is the obvious one of $\Omega(n)$ messages; while the best upper bound is $O(n+t^2)$ (cf. algorithms in [6, 18] which work under general omission failures and, *a fortiori*, under sending omission failures). Thus, it is still possible that a lower bound of $\Omega(n + t^2)$ can be established in this case. Unfortunately, Dolev and Reischuk's proof does not apply to this model of failures, as it uses, in a crucial manner, faulty processes which are not silent but are prevented from receiving messages — a behaviour which is possible under general omission failures, but impossible under sending omission failures only [1]. Closing the gap between the upper and lower bounds of the worst case message complexity in the case of crash and sending omission failures remains an open problem. Also, given the great sensitivity of performance on the number of messages, it would be useful to obtain exact, rather than asymptotic, bounds — as was the case for the problem studied in this paper.

[2] As this paper was in the process of being refereed Hadzilacos and Halpern [15] answered these questions by providing tight bounds on the worst and average case failure-free message complexity of binary BA protocols for all other types of failures that are considered in the literature on the BA problem. They also consider the case of non-binary BA; among other results, they provide a lower bound that matches the upper bound given above for non-binary BA in the case of crash failures.

[3] Dolev and Reischuk actually consider a failure model, called *arbitrary failures with message authentication*, where faulty processes are less restricted in their possible behaviour, but their lower bound proof applies verbatim even under the stronger assumption of general omission failures.

Acknowledgements

We are grateful to Joe Halpern for his comments on an earlier draft and several stimulating conversations, and the referees for their suggestions. This work was supported, in part, by the Natural Sciences and Engineering Research Council of Canada.

Bibliography

[1] Amdur, E.S. *On the Message Complexity of Byzantine Agreement.* Master's Thesis, Department of Computer Science, University of Toronto, September 1988.

[2] Berman, P., J. Garray and K. Perry. "Recursive Phase King Protocols for Distributed Consensus". Technical Report CS-89-24, Computer Science Department, Pennsylvania State University, August 1989.

[3] Bracha, G. Unpublished manuscript, Department of Computer Science, Cornell University, July 1984.

[4] Coan, B. and J. Welch. "A Byzantine Agreement Protocol with Optimal Message Bit Complexity". *Proc. of the 27th Annual Allerton Conference on Communication, Control and Computing.* pp. 1062–1071, September 1989.

[5] Cook, S.A., C. Dwork and R. Reischuk. "Upper and Lower Bounds for Parallel RAMS Without Simultaneous Writes". *SIAM Journal on Computing* 15(1):87–97, February 1986.

[6] Dolev, D. and R. Reischuk. "Bounds on Information Exchange for Byzantine Agreement". Research Report RJ 3587, IBM San Jose Research Laboratory, August 1982.

[7] Dolev, D. and H.R. Strong. "Authenticated Algorithms for Byzantine Agreement". *SIAM Journal on Computing* 12(4):656–666, November 1983.

[8] Dwork, C. and Y. Moses. "Knowledge and Common Knowledge in a Byzantine Environment I: Crash Failures." In *Proc. of the Conf. on Theoretical Aspects of Reasoning About Knowledge.* pp. 149–169, March 1986.

[9] Feldman, P. and S. Micali. "Byzantine Agreement in Constant Expected Time (and Trusting No One)". *Proc. of the 26th Annual IEEE Symp. on Foundations of Computer Science.* pp. 267–276, October 1985.

[10] Feldman, P. and S. Micali. "An Optimal Algorithm for Synchronous Byzantine Agreement". Technical Report MIT/LCS/TM-425, Laboratory for Computer Science, Massachusetts Institute of Technology, June 1990.

[11] Fischer, M.J. "The Consensus Problem in Unreliable Distributed Systems". Research Report RR-273, Department of Computer Science, Yale University, June 1983.

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

[13] Gray, J.N. "The Cost of Messages". In *Proc. of the 7th ACM Symp. on Principles of Distributed Computing.* pp. 1–7, August 1988.

[14] Hadzilacos, V. *Issues of Fault Tolerance in Concurrent Computations.* Ph.D. dissertation, Harvard University, June 1984.

[15] Hadzilacos, V. and J.Y. Halpern. "Message-Optimal Protocols for Byzantine Agreement". Forthcoming, 1990.

[16] Lynch, N.A. "A Hundred Impossibility Proofs for Distributed Computing". In *Proc. of the 8th ACM Symp. on Principles of Distributed Computing.* pp. 1–27, August 1989.

[17] Pease, M., R. Shostak and L. Lamport. "Reaching Agreement in the Presence of Faults". *Journal of the ACM* 27(2):228–234, April 1980.

[18] Weber, S.M. *Bounds on the Message Complexity of Byzantine Agreement.* Masters' Thesis, Department of Computer Science, University of Toronto, October 1989.