Time and Message Efficient Reliable Broadcasts

Tushar Chandra*
Sam Toueg*

TR 90-1094
May 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported by NSF Grant No. CCR-8901780.

Time and Message Efficient Reliable Broadcasts

Tushar Chandra* Sam Toueg*
Department of Computer Science
Upson Hall, Cornell University
Ithaca, New York 14853
e-mail(chandra@cs.cornell.edu, sam@cs.cornell.edu)

May 15, 1990

1 Introduction

Reliable Broadcast is a fundamental problem of fault-tolerant distributed computing. Informally, Reliable Broadcast requires that, despite failures, every message broadcast is consistently received by all the correct processes in the system. The following formulation of Reliable Broadcast, called Byzantine Agreement, has been extensively studied [PSL80,LSP82]. A process, the *general*, broadcasts a message m and all the processes attempt to reach agreement on the message broadcast. An algorithm solves Byzantine Agreement if it satisfies the following requirements[1]:

Validity: If the *general* is correct, all correct processes agree on m.

Agreement: All correct processes that decide must decide on the same value.

Eventual Decision: All correct processes eventually decide.

*Supported by NSF grant number CCR-8901780

[1]Note that Byzantine Agreement does not impose any restriction on the decision of faulty processes. In Section 9 we consider the Uniform Agreement problem which restricts the decision of faulty processes to be consistent with that of correct processes [Nei88,GT89].

Early solutions to the Byzantine Agreement problem were expensive in both time and message complexity. With these solutions, broadcasts complete in time proportional to t, where t is an a priori upper bound on the number of processes that could be faulty. To speed up broacasts, the concept of *early-stopping* was introduced by [DRS82]. Dolev *et al.*'s early-stopping algorithms ensure that broadcasts complete in time proportional to f, the number of processes that *actually* fail during execution.

However, known early-stopping algorithms are still expensive in terms of the number of messages they require [DR83,Had83,Per85,PT84,PT86,Ezh87]. For example, [PT86] describes an early-stopping algorithm for general-omission failures that stops by round $f+2$ but requires $O(fn^2)$ messages (where n is the total number of processes). Other researchers have concentrated on minimizing the number of messages exchanged [Web89,DS83,Bra82]. Such solutions have a poor time complexity. For example, the algorithm in [Bra82] tolerates upto t crash failures with only $O(n+t\sqrt{t})$ messages, but it requires $O(t)$ rounds.

In this paper, we consider *crash, send-omission*, and *general-omission* failures. For these models, we describe Reliable Broadcast algorithms that are efficient both in time and message complexity: broadcasts complete in $O(f)$ rounds using $O(fn)$ messages. With these algorithms, each additional process that fails can increase the cost of a broadcast by at most $O(n)$ messages and a constant number of rounds. In the case of general-omission failures, this is close to the lower bounds of $O(f)$ rounds and $O(ft)$ messages per broadcast.

2 Model and Definitions

We assume a system of n processes that can communicate through reliable links in a fully connected point-to-point network. Processes have unique *ids* in the range $[1, n]$ which are known a priori to all processes. The algorithm executes in synchronous rounds. Informally, a round is an interval of time where processes first send messages (according to their states), wait to receive messages sent by other processes in the same round, and then change their states accordingly. We consider the following types of process failures:

Crash failures: A faulty process fails by halting prematurely. Until it halts, it behaves correctly[2].

Send-omission failures: A faulty process may fail not only by crashing, but also by omitting to send some of the messages that it should send.

General-omission failures: A faulty process may fail by halting or by omitting to send or receive messages.

We define the notions of *decision, quiescence* and *termination* (in the context of agreement algorithms) as follows. A set of processes has *decided by time t* if all the members of this set have decided (irrevocably) by time t. A set of processes is said to be *quiescent at time t* if none of its members sends a message after time t[3]. A set of processes has *terminated* the execution of an algorithm *by time t* if all its members have halted or crashed by time t.

3 Outline of Algorithms

The algorithms in this paper use the rotating coordinator paradigm [CM84,Rei82]. A subset of $t+1$ processes cyclically become coordinators for a constant number of rounds each. The general is the first coordinator and its *id* is 1. When a process becomes a coordinator, it determines a "consistent" decision value and tries to impose it on the remaining processes. Our algorithms ensure that when a correct process becomes the coordinator, it will succeed in enforcing agreement. Since at most f coordinators can be faulty during an execution of the algorithm, agreement is achieved in $O(f)$ rounds. Moreover, in each round, most of the messages are to or from the coordinator; thus the number of messages sent is $O(n)$ per round, and agreement is reached with $O(fn)$ messages.

Each process p maintains a variable $estimate_p$ that represents p's current estimate of the final decision value. Processes can be in one of two states: *undecided* or *decided.*

---

[2]If a process p crashes in round r then any subset of events on p in round r could fail to occur. It is possible to weaken this assumption and derive better algorithms.

[3]Usually quiescence includes channel quiescence as well. In a round based system, channel quiescence is achieved at most one round after process quiescence.

A process p *decides* v when it sets $decision_p \leftarrow v$ and $state_p \leftarrow decided$. Our algorithms ensure that if a correct process p decides v (for some v), then all correct processes eventually decide v.

4 Reliable Broadcast for Crash Failures

Algorithm 1a tolerates crash failures. Each coordinator becomes "active" for three rounds. In the first round *undecided* processes send a request for "help" to the current coordinator c (An *undecided* coordinator "sends" a request to itself). If the current coordinator c does not receive any request, it skips rounds 2 and 3. If c receives a request, it broadcasts $estimate_c$ in round 2, and *decide* in round 3. Note that due to the crash failure assumption, if c begins to broadcast *decide*, then it must have successfully sent $estimate_c$ to all. Thus, *decide* is sent only if all processes receive $estimate_c$: all future coordinators are guaranteed to have the same message estimate, and will eventually force all processes to decide on it. The proof of correctness of algorithm 1a is as follows.

Let T be the round in which the first *decide* message is received by any process. Let p be the coordinator that sent this *decide*, and let $estimate_p$ be the message p broadcast in round $T-1$. We will show that all correct processes eventually decide $estimate_p$.

Lemma 1.1: At round $T-1$, all processes q which did not crash received $estimate_p$ and set $estimate_q \leftarrow estimate_p$.

Proof: Since p sent *decide* in round T, it did not crash in round $T-1$. Since p can only fail by crashing it must have sent $estimate_p$ to all processes in round $T-1$. Thus all processes q which did not crash by round $T-1$ receive $estimate_p$ and set $estimate_q \leftarrow estimate_p$. ∎

Lemma 1.2: If c is a coordinator after p, and c sends $estimate_c$, then $estimate_c = estimate_p$.

Proof: The proof follows from the previous lemma and an easy induction argument. ∎

Lemma 1.3: If coordinator c is correct, all processes which have not crashed decide by the end of round $3c$.

{*Initialization*}

$$estimate_p \leftarrow \begin{cases} m & \text{if } p \text{ is the general } (m \text{ is general's value}) \\ \perp & \text{Otherwise} \end{cases}$$

$state_p \leftarrow undecided$

{*End Initialisation*}

For $c \leftarrow 1, 2, ..., t+1$ **do**

{*Processor c becomes the coordinator for three rounds*}

Round 1: All *undecided* processes p send *request* to c
if c does not receive any *requests* then it skips rounds 2 and 3

Round 2: c broadcasts $estimate_c$
All *undecided* processes p that receive $estimate_c$ set $estimate_p \leftarrow estimate_c$

Round 3: c broadcasts *decide*
All *undecided* processes p that receive a *decide* do
$decision_p \leftarrow estimate_p$
$state_p \leftarrow decided$

od

Algorithm 1a: Reliable broadcast with crash failures

Proof: Suppose there is at least one undecided process q which has not crashed by the end of round $3c-2$. By the algorithm, q sends a request to c in round $3c-2$. As there are only crash failures, c receives this request. In round $3c-1$, c broadcasts $estimate_c$ which is received by all correct processes. Thus, in round $3c$, all the undecided processes which have not crashed receive *decide* from c. ∎

Theorem 1: Algorithm 1a solves Byzantine Agreement in the presence of crash failures. The correct processes decide and become quiescent by round $3f+3$ using $O(fn)$ messages.

Proof: It is easy to show that if the general is correct then all correct processes decide on the general's value in round 3. Hence the algorithm satisfies the *validity* condition. The *agreement* condition directly follows from Lemma 1.2. From the definition of f, one of the first $f+1$ coordinators is correct. Consequently from Lemma 1.3, all processes which have not crashed by the end of round $3f+3$ decide by that round. Thus *eventual decision* is achieved by round $3f+3$, and the algorithm solves Byzantine Agreement.

By Lemma 1.3, by the end of round $3f+3$, all processes have either decided or crashed. Since only undecided processes send requests to coordinators, and coordinators send messages only if they receive such requests, the whole system becomes quiescent by round $3f+3$. ∎

Note that the processes do not terminate the execution of the algorithm by round $3f+3$: the $t+1^{st}$ coordinator must wait for potential requests until round $3t+1$[4].

We can easily improve the time complexity of Algorithm 1a by merging rounds 1 and 2. In round 1, any process which has not decided sends the coordinator a *request.* Furthermore, if the coordinator c has not decided, it broadcasts $estimate_c$ (note that if c is decided at this point, then all surviving processes must have the same estimate as c). In round 2, c sends *decide* if it received a *request* in round 1. With this modification, the correct processes decide and become quiescent by round $2f+2$.

Further improvements are possible using pipelining. So far we only allowed a single coordinator to be active at a time. We can speed up the algorithm by pipelining its execution so that coordinator $i+1$ starts only one round after coordinator i (while i is still active). Thus coordinator c starts in round c. The resulting algorithm achieves

[4]However the algorithm can be modified to achieve early-termination at the cost of additional messages. See Section 8.

decision in $f+2$ rounds. See the Appendix for details.

5 Real-Time Reliable Broadcast for Crash Failures

Algorithm 1a relies on the following assumptions:

1. Execution proceeds in synchronous rounds.
2. All processes know a priori which process initiated a broadcast and in which round.
3. The identities and the order of all the coordinators is common knowledge.

These assumptions preclude the use of Algorithm 1a in a real-time environment where any process can initiate a broadcast at any time, and where dynamic failures prevent the use of a static agreement on the coordinators. We can overcome these limitations as follows:

- We replace Assumption 1 with the assumption that processes have synchronized clocks[5], and that communication delay is bounded by a constant δ.
- We remove Assumption 2. Any process can initiate a broadcast at any time. However, since there is no a priori knowledge of who broadcasts and when, correct processes may never become aware of a broadcast initiated by a process that crashes. Therefore, we must replace the *eventual decision* condition with:

  Uniform Decision: If any correct process decides, all correct processes decide.

  The resulting specification allows the correct processes to completely ignore a broadcast initiated by a faulty process. Similar specifications have been studied in [CASD82,GT89,CM84,SGS84].

[5]This is to simplify the presentation of the algorithm. However, approximately synchronized clocks are sufficient.

- We also remove Assumption 3. The initiator of a broadcast decides on a list of future coordinators, and includes this list in its initial broadcast[6]. This list is also piggybacked on subsequent messages related to this broadcast.

See Algorithm 1b.

6 Reliable Broadcast for Send-Omission Failures

Algorithms 1a and 1b do not tolerate send-omission failures. For example a faulty coordinator c could first omit to send $estimate_c$ to the next coordinator, and then send *decide* to one correct process p. Thus p decides on $estimate_c$ while the next coordinator, unaware of this estimate, can make undecided processes decide on $\perp$. This leads to disagreement. To correct this problem, we add an extra round in which processes that did not receive $estimate_c$ send a NACK to c. If c receives any NACK, it does not broadcast *decide*.

However, even with this modification, disagreement is possible. For example, a faulty coordinator c omits to send $estimate_c$ to a faulty process c' which fails to send a NACK to c. c does not receive any NACKs and thus proceeds to send some *decide*s. Then c' becomes the new coordinator without having received $estimate_c$. At this point it is possible that some correct process decided on $estimate_c$ while other correct processes are still undecided and rely on c' for a decision value.

To solve this problem, a request message from an undecided process p now includes $estimate_p$ with an associated *coordinator id.* This is the *id* of the coordinator that sent this estimate to p. An undecided coordinator c considers all the requests that it receives, and sets $estimate_c$ to the estimate with the largest associated coordinator id. See algorithm 2.

Let T be the round in which the first *decide* is received by any process. Let p be the coordinator that sent this *decide* and let $estimate_p$ be the message p broadcast in round $T-2$. We will show that all correct processes eventually decide $estimate_p$.

[6]Note that the initiator of a broadcast can decide the resiliency of that broadcast: the length of the list determines the maximum number of coordinators crashes that can be tolerated.

{*Initialization*}

$$(\textit{estimate}_p, \textit{coord-list}_p) \leftarrow \begin{cases} (m, \text{ list of } t \text{ processes}) & \text{if } p \text{ is the general} \\ (\perp, \perp) & \text{Otherwise} \end{cases}$$

$\textit{state}_p \leftarrow \textit{undecided}$

{*End Initialisation*}

If p is the general then

Broadcast $(\textit{estimate}_p, \textit{coord-list}_p, 0)$

Broadcast *decide*

cobegin

□ Upon the first receipt of an estimate by process p do {*Say estimate came from c*}

$(\textit{estimate}_p, \textit{coord-list}_p, \textit{coord-index}_p) \leftarrow (\textit{estimate}_c, \textit{coord-list}_c, \textit{coord-index}_c)$

$\textit{start-time}_p \leftarrow \textit{local-time}$

repeat at intervals of 3δ starting from $\textit{start-time}_p + 2\delta$

if $\textit{state}_p = \textit{decided}$ then **exit repeat**

else

$\textit{coord-index}_p \leftarrow \textit{coord-index}_p + 1$

send $(\textit{request}, \textit{estimate}_p, \textit{coord-list}_p, \textit{coord-index}_p)$ to $\textit{coord-list}_p[\textit{coord-index}_p]$

forever

□ Upon the first receipt of a *decide* do

$\textit{decision}_p \leftarrow \textit{estimate}_p$

$\textit{state}_p \leftarrow \textit{decided}$

□ Upon the first receipt of a *request* do {*Say request came from q*}

$(\textit{estimate}_p, \textit{coord-list}_p, \textit{coord-index}_p) \leftarrow (\textit{estimate}_q, \textit{coord-list}_q, \textit{coord-index}_q)$

Broadcast $(\textit{estimate}_p, \textit{coord-list}_p, \textit{coord-index}_p)$

Broadcast *decide*

coend

Algorithm 1b: Real time version of Algorithm 1a

{*Initialization*}

$$(estimate_p, coord\text{-}id_p) \leftarrow \begin{cases} (m, 0) & \text{if } p \text{ is the general } (m \text{ is general's value}) \\ (\perp, -1) & \text{Otherwise} \end{cases}$$

$state_p \leftarrow undecided$

{*End Initialisation*}

For $c \leftarrow 1, 2, ..., t+1$ **do**

{*Processor* c *becomes the coordinator for four rounds*}

Round 1: All *undecided* processes p send $(request, estimate_p, coord\text{-}id_p)$ to c
if c does not receive any *request* then it skips rounds 2 to 4
else
$estimate_c \leftarrow estimate_p$ with largest $coord\text{-}id_p$

Round 2: c broadcasts $(estimate_c, c)$
All *undecided* processes p that receive $(estimate_c, c)$ do
$(estimate_p, coord\text{-}id_p) \leftarrow (estimate_c, c)$

Round 3: All *undecided* processes p that did not receive $estimate_c$ in round 2
send NACK to c

Round 4: If c does not receive a NACK then c broadcasts *decide*
else c HALTS {*The coordinator detects its own failure*}
All *undecided* processes p that receive *decide* do
$decision_p \leftarrow estimate_p$
$state_p \leftarrow decided$

od

Algorithm 2: Reliable broadcast with send-omission failures

Lemma 2.1: By the end of round $T-2$, all correct processes q receive $estimate_p$ and set $estimate_q \leftarrow estimate_p$.

Proof: Suppose, for contradiction, some correct process q does not receive $estimate_p$ by the end of round $T-2$. By the definition of T, q must be undecided in round $T-1$. Thus q sends a NACK to p in round $T-1$. Since q is correct, and only send-omission failures can occur, p receives this NACK and does not send *decide* in round T - a contradiction. ∎

Lemma 2.2: Suppose a correct process q sets ($estimate_q$, $coord\text{-}id_q$) in "round 2" of the algorithm to some value (v, r). Then, until q decides, all the coordinators can only send v as their estimate.

Proof: The proof is by induction. Its is clear that coordinator r can only send v as its estimate. Suppose that when c becomes coordinator, q is still undecided. By the induction hypothesis, coordinators from r to $c-1$ can only send v as their estimate value. We now show that coordinator c can only send v as its estimate. Since coordinators r through $c-1$ can only send v as their estimate, any estimate associated with a coordinator id $\in [r, c-1]$ must also have value v. When c becomes the coordinator, all undecided processes p (including q) send ($request$, $estimate_p$, $coord\text{-}id_p$) to c. The induction hypothesis implies that if $coord\text{-}id_p \geq r$ then $estimate_p = v$. Since $coord\text{-}id_q \geq r$, c will set $estimate_c \leftarrow v$. ∎

Lemma 2.3: If coordinator c is correct, all correct processes decide by the end of round $4c$.

Proof: Similar to Lemma 1.3. ∎

Lemma 2.4: All correct processes which decide must decide on the same value.

Proof: From Lemma 2.1, at the end of round $T-2$ all correct processes get the same estimate say v. From Lemma 2.2, it follows that any correct process that decides can only decide v. ∎

Theorem 2: Algorithm 2 solves Byzantine Agreement in the presence of send-omission failures. The correct processes decide by round $4f+4$ using $O(fn)$ messages.

Proof: It is easy to show that if the general is correct then all correct processes decide on the general's value in round 4. Hence the algorithm satisfies the *validity* condition. The *agreement* condition directly follows from Lemma 2.4. From the definition of f, one of the first $f+1$ coordinators is correct. Consequently from Lemma 2.3, all correct processes decide by round $4f+4$. Thus *eventual decision* is achieved by round $4f+4$, and the algorithm solves Byzantine Agreement. ∎

7 Reliable Broadcast for General-Omission Failures

Algorithm 2 tolerates any number of send-omission failures (i.e., for any $t < n$). To tolerate general-omission failures we used a "translation" technique from [NT90] which requires $n > 2t$. Thus, the resulting algorithm tolerates upto $\lfloor (n-1)/2 \rfloor$ general-omission failures. Informally, running algorithm 2 in a system with general-omission failures does not work for the following three reasons:

1. A faulty coordinator could fail to receive a NACK, and thereby send a *decide* when it should not. This problem is remedied by the translation mechanism: essentially the NACK mechanism is replaced with $n-t$ positive ACKs.

2. A faulty coordinator that is activated by a $(\mathit{request}, \mathit{estimate}_p, \mathit{coord\text{-}id}_p)$ may fail to receive a $(\mathit{request}, \mathit{estimate}_q, \mathit{coord\text{-}id}_q)$ with $\mathit{coord\text{-}id}_p < \mathit{coord\text{-}id}_q$, where q is a correct process. To solve this problem, an activated coordinator c broadcasts a *probe* asking all processes p to send $(\mathit{estimate}_p, \mathit{coord\text{-}id}_p)$. The coordinator must receive at least $n-t$ responses before it updates $\mathit{estimate}_c$ and broadcasts it.

3. A faulty process may continuously fail to receive *decide* messages and thus successively send *requests* to all coordinators, thereby activating all of them. This results in too many messages. To overcome this problem, we introduce a technique that prevents a faulty process from activating more than one correct coordinator. So at most $2f+1$ coordinators will be activated, resulting in $O(fn)$ messages. The technique works as follows. An activated coordinator c selects one of the processes which woke it up, called the *requester*. Any process p that decides, relays its decision value to the *requester*. If later p becomes a coordinator, it ignores any request from this *requester*.

With Algorithm 3 a single coordinator executes every 7 rounds. This can be improved by pipelining (see the Appendix for details).

{*Initialization*}

$$(estimate_p, coord\text{-}id_p) \leftarrow \begin{cases} (m, 0) & \text{if } p \text{ is the general } (m \text{ is general's value}) \\ (\perp, -1) & \text{Otherwise} \end{cases}$$

$state_p \leftarrow undecided$

$finishedset_p \leftarrow \Phi$

{*End Initialisation*}

For $c \leftarrow 1, 2, ..., t+1$ **do**

{*Processor c becomes the coordinator for seven rounds*}

Round 1: All *undecided* processes p send *request* to c
  Let $Q_c = \{q \mid c \text{ received a request from } q \wedge q \notin finishedset_c\}$
  If $Q_c = \Phi$ then c skips rounds 2 to 7
  else $requester \leftarrow$ an element of Q_c

Round 2: c broadcasts *probe*

Round 3: All processes p that receive a *probe* send $(answer, estimate_p, coord\text{-}id_p)$ to c

Round 4: If c receives $\geq n - t$ answers then
    $estimate_c \leftarrow estimate_p$ with largest $coord\text{-}id_p$
    c broadcasts $(estimate_c, c)$
  else c HALTS {*The coordinator detects its own failure*}
  All *undecided* processes p that receive $(estimate_c, c)$ do
    $(estimate_p, coord\text{-}id_p) \leftarrow (estimate_c, c)$

Round 5: All processes p that received $(estimate_c, c)$ send an ACK to c

Round 6: If c receives $\geq n - t$ ACKs then c broadcasts $(decide, requester)$
  else c HALTS {*The coordinator detects its own failure*}
  All *undecided* processes p which received $estimate_c$ and *decide* do
    $decision_p \leftarrow estimate_p$
    $state_p \leftarrow decided$

Round 7: All processes p that received $estimate_c$ and *decide* do
    Add *requester* to $finishedset_p$
    send $(decide, estimate_p)$ to *requester*
  If *requester* is *undecided* and it receives $(decide, estimate_p)$ for some p
    $decision_{requester} \leftarrow estimate_p$
    $state_{requester} \leftarrow decided$

od

Algorithm 3: Reliable broadcast with general-omission failures

If $Q_c \neq \Phi$, we say that coordinator c is active. Let T be the round in which the first *decide* is received by any process. Let p be the coordinator that sent this *decide* and let $estimate_p$ be the message p broadcast in round $T-2$. We will show that all correct processes eventually decide $estimate_p$.

Lemma 3.1: By the end of round $T-2$, at least $n-t$ processes q receive $estimate_p$ and set $estimate_q \leftarrow estimate_p$.

Proof: For p to broadcast *decide* in round k, it must receive $n-t$ ACKs in round $k-1$ thus at least $n-t$ processes receive $estimate_p$ and set $estimate_q \leftarrow estimate_p$. ∎

Lemma 3.2: If $t+1$ processes receive $estimate_c$ from coordinator c, all future coordinators which send out their estimate, send out $estimate_c$.

Proof: The proof, is by induction and is similar to Lemma 2.2. Assume that all coordinators $\in [c, c'-1]$ that send out their estimate, send out $estimate_c$. Thus all we have to show is that if c' broadcasts $estimate_{c'}$ then it previously received an answer from some process p for which $coord\text{-}id_p \geq c$. From the algorithm it is clear that if c' broadcasts $estimate_{c'}$ then it must have received *answers* from at least $n-t$ processes. Hence it must have received an *answer* from one of the $t+1$ processes which received $estimate_c$ from c. For any such process q, $coord\text{-}id_q \geq c$. ∎

Lemma 3.3: If coordinator c is correct, all correct processes decide by the end of round $7c-1$.

Proof: Similar to Lemma 2.3. ∎

Lemma 3.4: All processes which decide, decide on the same value.

Proof: From Lemma 3.1, 3.2 and the fact that $n-t \geq t+1$. ∎

Lemma 3.5: At most $f+1$ correct coordinators become active.

Proof: When the first correct coordinator becomes active, all correct processes decide and will never be *requesters* in the future. So only faulty processes can activate additional correct coordinators.

We now show that a faulty process can activate at most one correct coordinator. Suppose r is faulty and activates a correct coordinator c (r is the *requester* c chooses). c will ensure that all correct processes include r in their *finishedsets*. Thus r cannot activate any future correct coordinator. ∎

Theorem 3: Algorithm 3 solves Byzantine Agreement in the presence of general-omission failures. The correct processes decide by round $7f+6$ using $O(fn)$ messages.
Proof: It is easy to show that if the general is correct then all correct processes decide on the general's value in round 6. Hence the algorithm satisfies the *validity* condition. The *agreement* condition directly follows from Lemma 3.4. From the definition of f, one of the first $f+1$ coordinators is correct. Consequently from Lemma 3.3, all correct processes decide by round $7f+6$.

It is easy to see that when the current coordinator is correct but not active, correct processes do not send any messages. Let $c_{correct}$ be the number of correct processes which become active coordinators and c_{faulty} be the number of faulty coordinators. From Lemma 3.5, $c_{correct} \leq f+1$. The number of messages sent by correct coordinators is bound by:

$$c_{correct}O(n) \leq O(fn) \text{ messages}$$

The number of messages sent by the $n-f$ correct processes while they are not coordinators is bound by:

$$(n-f)*(c_{correct}+c_{faulty})*O(1) \leq O(fn) \text{ messages}$$

Thus the total number of messages sent by correct processes is bound by $O(fn)$. Hence algorithm 3 solves Byzantine Agreement using $O(fn)$ messages. ∎

Note that the total number of messages sent by the f faulty processes is also $O(fn)$.

Theorem 4: Any algorithm which solves the Byzantine Agreement problem with general-omission failures requires $O(ft)$ messages in the worst case.
Proof: A minor extention of a proof in [DR83][7].

8 Termination at a Cost

Even though Algorithms 1a, 1b, 2 and 3 achieve decision in $O(f)$ rounds, processes can take $O(t)$ rounds to halt. However, all these algorithms can be modified so that

[7]The result in [DR83] shows that any algorithm which solves the Byzantine Agreement problem with general-omission failures requires $O(t^2)$ messages in the worst case.

{*Initialization*}
$k_p \leftarrow t$
{*End Initialisation*}
The current coordinator c executes the following $\lceil \log_2 n \rceil$ rounds:

For $i \leftarrow 4$ to $\lceil \log_2 n \rceil + 3$ **do**
Round i: c sends *stop* to processes with $id \in [k_c/2, k_c]$
$k_c \leftarrow k_c/2$
All processes p receiving *stop* **halt**
od

Algorithm 4: Achieving early-termination for crash failures

some (or all) processes halt as soon as they decide, at the cost of n messages for every process that halts early. In particular, all processes can halt in $O(f)$ rounds at the cost $O(n^2)$ messages. To do so, any deciding process broadcasts the decision to all processes and then halts[8].

For crash and send-omission failures, we can refine this idea to achieve termination in $O(f + \log t)$ rounds with $O((f + \log t)n)$ messages. For crash failures, this is done by first appending Algorithm 4 to Algorithm 1a, and then pipelining the execution (as shown in the Appendix) so that $O(\log t)$ coordinators run simultaneously. A similar modification applied to Algorithm 2, appending Algorithm 5 and pipelining, achieves the same result for send-omission failures.

9 Uniform Agreement

Byzantine Agreement does not impose any restriction on the behaviour of faulty processes. However, for "benign" failures such as omission failures (where processes do not change state arbitrarily or lie), we can require that the state of faulty processes satisfy some requirements. *Uniform Agreement* is the Byzantine Agreement problem with the additional requirement that all processes that reach a decision, including the faulty

[8]For crash and send-omission failures, we can improve this to $O(f)$ rounds and $O(fn + t^2)$ messages.

{*Initialization*}
$k_p \leftarrow t$
{*End Initialisation*}
The current coordinator c executes the following $2\lceil \log_2 n \rceil$ rounds:

Round 5: All *undecided* processes p send NACK to c
If c receives a NACK then c **halts** {*The coordinator detects its own failure*}
For $i \leftarrow 3$ to $\lceil \log_2 n \rceil + 2$ **do**
Round $2i$: c sends *stop* to processes with $id \in [k_c/2, k_c]$
All processes p receiving *stop* **halt**
Round $2i + 1$: All processes $\in [k_c/2, k_c]$ (which have not halted) do:
Send NACK to c
If c receives a NACK then c **halts** {*The coordinator detects its own failure*}
else $k_c \leftarrow k_c/2$ **od**

Algorithm 5: Achieving early-termination for send-omission failures

ones, decide on the same value. Thus Uniform Agreement strengthens the *agreement* condition to:

Uniformity: All processes that decide, decide on the same value.

We can show that Algorithms 1 and 3 actually solve Uniform Agreement, while the algorithm for send-omission failures does not. It can be shown that Uniform Agreement in a system with general-omission failures requires $n > 2t$ [NT90]. Algorithm 3 matches this bound, and so it is optimal in the number of faulty processes it tolerates.

Appendix - Pipelining

It is possible to speed up all the algorithms in this paper by a constant factor. This is achieved by pipelining their executions. In the pipelined versions, in each round there are many active coordinators - each one at a different stage of the algorithm. A brief description of this pipelining scheme and its performance follows:

Crash failures: When process i begins its second round as coordinator, process $i+1$ begins its first round. It can be shown that the correctness of the algorithm is preserved. The system decides by round $f+2$ and is quiescent by round $f+3$.

Send-omission failures: Coordinator $i+1$ starts when coordinator i begins its third round. With this, the system decides in $2f+4$ rounds. With a few additional modifications, decision can be achieved in $2f+1$ rounds for Byzantine Agreement and in $2f+3$ rounds for Uniform Agreement.

General-omission failures: As in the send-omission case, two successive coordinators can be run with a gap of two rounds between them. With this modification decision is achieved in $2f+6$ rounds. With a few additional changes decision can be achieved in $2f+3$ rounds.

Acknowledgements

We would like to thank Navin Budhiraja, Ajei Gopal, Prasad Jayanti, Keith Marzullo, Pat Stephenson and the distributed systems group at Cornell University for their critical comments. Navin Budhiraja proposed a simplified version of Algorithm 1a which took $O(n^2)$ messages to reach agreement.

References

[Bra82] Gabriel Bracha. Personal communication. 1982.

[CASD82] Flaviu Cristian, Houtan Aghili, H. Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. pages 200–206, 1982. A revised version appears as IBM Technical Report RJ5244.

[CM84] J. Chang and N. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[DR83] Danny Dolev and R. Reischuk. Bounds on information exchange for Byzantine agreement. Technical report, IBM Research Laboratory, IBM Research Laboratory, San Jose, CA, 1983.

[DRS82] Danny Dolev, R. Reischuk, and Raymond Strong. 'Eventual' is earlier than 'Immediate'. In *Proceedings 23rd Symposium on Foundations of Computer Science, Chicago, Illinois*, pages 196–203, November 1982.

[DS83] C. Dwork and D. Skeen. The inherent cost of nonblocking commitment. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 1–11, August 1983.

[Ezh87] Paul D. Ezhilchelvan. Early stopping algorithms for distributed agreement under fail-stop, omission, and timing fault types. In *IEEE 1987 Sixth Symposium on Reliability in Distributed Software and Database Systems*, pages 201–212, Computing Laboratory, The university, Newcastle upon Tyne, England, 1987. IEEE computer society press.

[GT89] Ajei Gopal and Sam Toueg. Reliable broadcast in synchronous and asynchronous environments. In J.-C. Bermond and M. Raynal, editors, *Distributed Algorithms, 3rd International Workshop, Nice, France*, pages 110–123. Springer, September 1989.

[Had83] Vassos Hadzilacos. Byzantine agreement under restricted types of failures (not telling the truth is different from telling lies). Technical Report 18-83, Department of Computer Science, Harvard University, 1983.

[LSP82] Leslie Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Nei88] Gil Neiger. *Techniques for Simplifying the Design of Distributed Systems*. PhD thesis, Cornell University, August 1988. Department of Computer Science Technical Report 88-933.

[NT90] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed algorithms. Technical Report TR 90-1081, Cornell University, January 1990. To be published in the Journal of Algorithms.

[Per85] Kenneth J. Perry. *Early Stopping Protocols for Fault-Tolerant Distributed Agreement.* PhD thesis, Cornell University, February 1985. Department of Computer Science Technical Report 85-662.

[PSL80] M. Pease, R. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[PT84] K. J. Perry and Sam Toueg. An authenticated Byzantine generals algorithm with early stopping. Technical Report 84-620, Department of Computer Science, Cornell University, June 1984.

[PT86] Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, March 1986.

[Rei82] R. Reischuk. A new solution for the Byzantine general's problem. Technical report, IBM Research Laboratory, November 1982.

[SGS84] F. Schneider, D. Gries, and R. D. Schlichting. Fault-tolerant broadcasts. *Science of Computer Programming*, pages 1–15, 1984.

[Web89] Samuel Weber. Bounds on the message complexity of Byzantine agreement. Master's thesis, University of Toronto, October 1989.