Distributed Agreement in the Presence of Processor and Communication Faults*

Kenneth J. Perry
Sam Toueg
TR 84-610
May 1984
(Revised June 1985)

Department of Computer Science
Cornell University
Ithaca, New York 14853

* partial support for this work was provided by the National Science Foundation under grant MCS 83-03135.

To appear in: IEEE Transactions of Software Engineering, March 1986

Distributed Agreement in the Presence of Processor and Communication Faults

Kenneth J. Perry
Sam Toueg

Department of Computer Science
Cornell University
Ithaca, New York 14853

ABSTRACT

A model of distributed computation is proposed in which processes may fail by not sending or receiving the messages specified by a protocol. The solution to the Byzantine Generals Problem for this model is presented. Our algorithm exhibits early-stopping under conditions of less than maximum failure and is as efficient as the algorithms developed for the more restrictive crash-fault model in terms of time, message, and bit complexity. We show extant models to under-estimate resiliency when faults in the communication medium are considered; the model of this paper is more accurate in this regard.

1. Introduction

Given is a collection of n distributed, potentially faulty processes able to communicate only by messages. Desired is a protocol which allows all processes in the collection to agree on a value **m** chosen by a distinguished process *General*. The desired protocol is said to solve the Byzantine Generals Problem if it satisfies the constraints:

(Validity):

If *General* does not fail then all non-faulty processes agree on **m**.

(Agreement):

All non-faulty processes agree on a common value.

The Byzantine Generals Problem has been explored extensively, under varying types of failures[Cris84], [DS83], [FL82], [LF82], [LSP82], [Per84a], [Rabi83], [Toue84], [TPS85]. In the most restrictive model of failure, faulty processes may only `crash`, that

is, a process fails only by halting. Malicious failures are the least restrictive. In this model failed processes may not only deviate from the protocol but actively collude with one another in an attempt to prevent agreement.

Experience dictates that process failures are rarely of the type associated with either of the extremes described above. Previous work has attempted to bring the crash-fault model in greater conformity with reality by broadening the types of process failure. [Hadz83] allows processes to continue executing following failure; his only restriction is that a faulty process sends no false (relative to the protocol) messages. However, a faulty process may omit sending messages at will. In [Hadz83] the new failure mode is translated into crash-fault behavior at the cost of additional bit complexity.

This paper lessens the failure restrictions yet again. Processes in our model may fail not only in the sending of messages but in the *receiving* as well. Stated another way, a faulty process can halt prematurely, or fail to send or receive any message required by the protocol. Clearly we have the seeds of malicious behavior: a faulty process is now able to generate messages independent of those transmitted by the correct processes. Therefore we impose restrictions: a failed process may refuse to send or receive any message called for by the protocol; however, any message transmitted must be correct relative to the protocol and the set of messages accepted. We shall make these restrictions more precise in a subsequent section.

There are several reasons for pursuing this model of failure. First, it is another step toward a more realistic model of process failure. Additionally, most existing protocols assume that no messages are lost. This assumption is commonly rationalized by

declaring that a missing message may be accounted for by arbitrarily attributing the loss to a fault of either the sender or receiver. We will show that doing so severely underestimates the resiliency of the collection of processes. In our model, the fault may be laid squarely on the shoulders of the process committing the fault: When a message specified by the protocol is not successfully transmitted and received it is either because a faulty sender did not transmit it or a faulty receiver failed to receive it. (For example, in practice a message can be lost because the sender was tardy or the receiver had no available buffer). Thus, our model is an extension of that proposed by [Hadz83] (which includes crash-fault behavior as well). Moreover, our solutions are as efficient in time, message, and bit complexity as those previously proposed for the most restricted type of failure (crash-faults). Hence our results demonstrate that less restrictive failure assumptions do not always entail greater expense.

2. The model

Let P be the set $\{P_1, P_2, \cdots, P_n\}$ of processes participating in the protocol. Assume that a distinguished process $General \in P$ wants to transmit a binary value **m** to all others. (Efficient methods for converting a binary-valued Byzantine Generals protocol to one which works for arbitrary values may be found in [Per84b] and [TC84]).

Define a *phase* to be the interval of time in which each non-faulty process is able to exchange a message with all other non-faulty processes. Most algorithms for the Byzantine Generals Problem, ours included, are described by the actions taken in a sequence of phases.

Let $sent_i^R(j)$ denote the message sent by process P_i to P_j in phase R; $received_i^R(j)$ will be used to denote the message received by P_i from process P_j in phase R. Due to our model of failure, it is not necessarily the case that

$$sent_i^R(j) = received_j^R(i)$$

Define V_i^R, the *view of P_i through phase R* to be the collection of messages received by process P_i through the end of phase R:

$$V_i^0 = \phi$$

$$V_i^R = \{\langle received_i^{R'}(j), P_j, R' \rangle \,|\, P_j \in P, \text{ in phase } 1 \leq R' \leq R\}$$

A *protocol* Φ is a collection of functions, one per process, which map the view of a process in each phase into messages to be sent. Let $\Phi_i^R(j)$ represent the function which maps P_i's view through phase $R-1$ into the message which Φ dictates P_i send to P_j in phase R. We use $\mu_i^R(j)$ to denote this message:

$$\mu_i^R(j) = \left(\Phi_i^R(j)\right)(V_i^{(R-1)})$$

We say that process P_i *commits a send-fault with respect to process P_j in phase R* if $sent_i^R(j) \neq \mu_i^R(j)$. Similarly, P_i is said to *commit a receive-fault with respect to process P_j in phase R* if $received_i^R(j) \neq sent_j^R(i)$.

If processes were allowed to commit arbitrary send and receive faults, our model would encompass completely malicious behavior. For example, by committing receive-faults with respect to all processes in every phase, process P_i would be able to act without regard to the desires of the other processes (as communicated by their messages). Likewise may be said if P_i may commit malicious send-faults. We therefore impose the following restrictions on faulty behavior:

(Send-fault Restriction):

$$sent_i^R(j) \neq \mu_i^R(j) \Rightarrow sent_i^R(j) = \phi$$

(Receive-fault Restriction):

$$received_i^R(j) \neq sent_j^R(i) \Rightarrow received_i^R(j) = \phi$$

In other words, a process may refuse to send or receive any message; however, only ϕ (denoting the missing message) may be substituted for a proper message.

Let $FAULTY\text{-}SENDERS^R$ be the set of processes which have committed send-faults in any phase up to and including R. $FAULTY\text{-}RECEIVERS^R$ is analogously defined for receive-faults. That is

$$FAULTY\text{-}SENDERS^R = \{P_j \in P \mid sent_j^{R'}(i) \neq \mu_j^{R'}(i) \text{ for some } P_i \text{ in phase } R' \leq R\}$$

$$FAULTY\text{-}RECEIVERS^R = \{P_j \in P \mid received_j^{R'}(i) \neq sent_i^{R'}(j) \text{ for some } P_i \text{ in phase } R' \leq R\}$$

Process P_i is $faulty$ if there is some phase R in which

$$P_i \in FAULTY\text{-}SENDERS^R \cup FAULTY\text{-}RECEIVERS^R$$

The ability of a faulty process to deviate from its protocol is what makes Byzantine Agreement difficult to achieve. We bound the number of faulty processes by a constant, $t < (n-1)$.

Like most other work in this area, we shall assume that the processes communicate through a completely-connected, perfectly reliable message system in which all messages are correctly delivered. Processes are the only source of failure in our model. Thus, if in phase R process P_i commits no send fault with respect to P_j and if P_j commits no receive fault with respect to P_i then

$$sent_i^R(j) = \mu_i^R(j) = received_j^R(i)$$

In light of our model of process failures the message system reliability assumption is not unduly restrictive. Suppose a phase R message from process P_i to process P_j is not delivered due to a failure of the message system. We can model the failure as follows: if $sent_i^R(j) \neq \phi$ then interpret the lost message as a fault on the part of P_j; otherwise as a fault of the sender. Thus the fault is properly attributed to its source.

Finally, we say that process P_i has $decided$ as soon as $decision_i \in \{0, 1, \text{NIL}\}$.

3. The basic algorithm

A $(t + 1)$ phase algorithm which solves the Byzantine Generals Problem is presented in Figure 3.1. The basic idea is as follows: since processes are not malicious, any value received must be **m**. Hence, upon first receiving a value, process P_i decides and relays **m** to all other processes. Should no value be received by the end of the last phase, a process decides on NIL, indicating that *General* failed.

3.1. Correctness of the basic algorithm

Correctness of the basic algorithm is proved in this section. We begin by introducing some definitions which facilitate the proof.

Define $QUIET_i^R$ to be the set of processes from whom P_i has failed to receive a message in some phase up to and including R. Formally we define $QUIET_i^R$ as follows: let ES^R denote those processes eligible to send messages in phase R. This is simply

$ES^1 = \{General\}$

$ES^R = P$ for $R > 1$

Phase 1 (*General*):

$decision_{General} := \mathbf{m}$;
send **m** to all processes ;
halt

Process P_i after receiving all messages sent in phase R, $1 \leq R \leq (t+1)$:

if $decision_i = \text{'?'} \wedge \exists P_j \in P : received_i^R(j) \in \{0, 1\} \rightarrow$
  $decision_i := received_i^R(j)$
  if $R < t+1 \rightarrow$ **send** $received_i^R(j)$ to all processes in phase $R+1$
  $[]R = t+1 \rightarrow$ **skip**
  fi

$[]\, decision_i = \text{'?'} \wedge \nexists P_j \in P : received_i^R(j) \in \{0, 1\} \rightarrow$
  if $R < t+1 \rightarrow$ **send** '?' to all processes in phase $R+1$
  $[]R = t+1 \rightarrow decision_i := \text{NIL}$
  fi

$[]\, decision_i \neq \text{'?'} \rightarrow$ **skip**
fi

Figure 3.1 The basic algorithm (without early stopping)

Then

$$QUIET_i^R = \{P_j \in P \mid P_j \in ES^{R'} \wedge received_i^{R'}(j) = \phi \text{ for some } R' \leq R\}$$

This is the set of processes which P_i perceives to be faulty. (If P_i is non-faulty, this set in fact contains only faulty processes.)

Let $FAULTY\text{-}RELAYERS^R$ for $R < (t+1)$ be the set of processes which have received **m** and failed to relay it to all other processes by the end of phase R. That is

$FAULTY\text{-}RELAYERS^R =$

$$\{P_j \in P \mid \exists P_i, P_k \in P : received_j^{R'}(i) = \mathbf{m} \wedge sent_j^{(R'+1)}(k) \neq \mathbf{m} \text{ for phase } R' < R\}$$

By definition $received_{General}^0(General) = \mathbf{m}$. This makes *General* a member of

FAULTY-RELAYERS[1] if it does not complete its initial broadcast without failing. Observe that

$$FAULTY\text{-}RELAYERS^R \subseteq FAULTY\text{-}SENDERS^R$$

The conditions under which one process decides on **m** while another remains undecided are the subject of the next lemma. It states the number of failures which must occur in order for this situation to arise. This lemma will be invoked repeatedly when we determine whether **m** is still a potential decision value for an undecided process.

Lemma 3.1

Consider the system of processes at the end of phase $R < (t+1)$. Suppose there is an undecided process $P_i \notin FAULTY\text{-}RECEIVERS^R$. If there is a process P_j which first receives **m** in phase R then $|FAULTY\text{-}RELAYERS^R| \geq R$ and $FAULTY\text{-}RELAYERS^R \subseteq QUIET_k^R$ for all P_k which are still undecided.

Proof

By induction on R.

(Basis): $R = 1$

Since undecided process P_i has not committed a receive-fault, *General* must not have sent it a message. Hence $General \in FAULTY\text{-}RELAYERS^R$.

Any process P_k which has not reached a decision by the end of the first phase must have received no message from *General*. Thus $General \in QUIET_k^R$.

(Induction): Assume statement is true for all $R' < R$.

Let P_j be a process which first receives **m** in phase R. There must be a pro-

cess P_l such that $received_j^R(l) = \mathbf{m}$ but $sent_l^R(i) = \phi$. Therefore P_l first becomes a member of $FAULTY\text{-}RELAYERS^R$ in phase R. By the induction hypothesis $|FAULTY\text{-}RELAYERS^{(R-1)}| \geq R-1$; thus the addition of P_l results in $|FAULTY\text{-}RELAYERS^R| \geq R$.

For any process P_k which remains undecided by the end of phase R, $received_k^R(l) = \phi$ so $P_l \in QUIET_k^R$. Moreover, P_k receives no message from any process which first becomes a member of $FAULTY\text{-}RELAYERS^R$ in phase R. Hence all these processes are also included in $QUIET_k^R$. By the induction hypothesis, $FAULTY\text{-}RELAYERS^{(R-1)} \subseteq QUIET_k^{(R-1)}$. Therefore, $FAULTY\text{-}RELAYERS^R \subseteq QUIET_k^R$. □

We now prove the correctness of the algorithm of Figure 3.1.

Theorem 3.2

Consider a collection of n processes of which at most t commit send-faults. Let $n > (t+1)$. The algorithm of Figure 3.1 correctly solves the Byzantine Generals Problem.

Proof

(Validity):

If *General* is non-faulty then all $P_i \notin FAULTY\text{-}RECEIVERS^1$ agree on $\mathbf{m}$ after one phase.

(Agreement):

Because a faulty process may not generate arbitrary values, it is clear that $decision_i \in \{0, 1\}$ implies $decision_i = \mathbf{m}$. Therefore should the decisions of any

two non-faulty processes differ, it must be the case that one of the values is NIL. It is now demonstrated that if any non-faulty process P_i decides on **m** then all $P_j \in FAULTY\text{-}RECEIVERS^{(t+1)}$ reach the same decision.

Suppose P_i decides on **m** at the end of phase R. If $R < (t+1)$ then P_i sends **m** to all processes in phase $(R+1)$. Hence an undecided process which does not commit a receive-fault (with respect to P_i) in phase $(R+1)$ also decides on **m**. Since no process decides on NIL before the end of phase $(t+1)$, there are no processes which have decided on NIL by phase R.

For the case when $R = (t+1)$, let P_j be the process that sent **m** to P_i. Then P_j must first have decided on **m** in phase t. We invoke Lemma 3.1 which tells us that there must have been at least t processes in $FAULTY\text{-}RELAYERS^t$ (and hence also in $FAULTY\text{-}SENDERS^t$). By assumption this is the maximum number of send-faults which may occur. Therefore P_j commits no send fault in phase $(t+1)$. Hence all undecided $P_k \in FAULTY\text{-}RECEIVERS^{(t+1)}$ receive **m** from P_j and reach the same decision as P_i.

□

Observe that no limit was placed on the number of receive-faults. Thus it can be seen that any model in which receive-faults count in the number of faults tolerated under-estimates resiliency to failure. Stated another way, our algorithm renders receive-faults benign, therefore their number need not be bounded.

It is easy to explain why only send-faults are relevant. Suppose some process P_j commits a receive fault with respect to P_i in phase R. Consider the message $sent_i^R(j)$ which is missed. If $sent_i^R(j) = \phi$ then P_j has unwittingly followed the protocol. If instead $sent_i^R(j) = \mathbf{m}$ then there are two cases to review. When the sender P_i commits no send-fault, all non-faulty processes receive and decide on $\mathbf{m}$; hence agreement is achieved in spite of P_j's failure. If the sender's transmission is faulty, let us just assume that $sent_i^R(j) = \phi$ and the above proof is still correct. We have thus shown our algorithm to be impervious to receive-faults.

Note that not all algorithms that are resilient to send-faults are able to tolerate receive-faults as well. For example, the early-stopping algorithm of [Hadz83] does not extend to receive-faults. A simple extension to our algorithm which exhibits early-stopping if fewer than t send-faults actually occur is the topic of the next section.

4. Early Stopping

The algorithm of Figure 3.1 is modified in order to permit processes to halt immediately after deciding. We will show that if $f < t$ send-faults take place, then all processes will have halted by the end of phase $(f + 2)$. Thus, the cost (in number of phases) of fault-tolerance is in direct proportion to the number of send-faults which *actually* occur. The early-stopping protocol is given in Figure 4.1.

4.1. Correctness of the Early Stopping Algorithm

We begin by observing that Lemma 3.1, whose correctness was proved for the previous protocol, applies to the new one as well. With the following lemma we show that

Phase 1 (*General*):

$decision_{General} := \mathbf{m}$;
send **m** to all processes ;
halt

Process P_i **after receiving all messages sent in phase** R, $1 \leq R \leq (t+1)$:

$QUIET_i^R := QUIET_i^{(R-1)} \cup \{P_j \in P \mid P_j \in ES^R \wedge received_i^R(j) = \phi\}$;

if $decision_i = \text{'?'} \wedge \exists P_j \in P : received_i^R(j) \in \{0, 1, \text{NIL}\} \rightarrow$
  $decision_i := received_i^R(j)$;
  if $R < t+1 \rightarrow$ **send** $received_i^R(j)$ to all processes in phase $R+1$
    $[]R = t+1 \rightarrow$ **skip**
  fi

$[]\, decision_i = \text{'?'} \wedge \nexists P_j \in P : received_i^R(j) \in \{0, 1, \text{NIL}\} \rightarrow$
  if $R < t+1 \rightarrow$
    if $R \leq |QUIET_i^R| \rightarrow$ **send** '?' to all processes in phase $R+1$
      $[]R > |QUIET_i^R| \rightarrow decision_i := \text{NIL}$;
        send NIL to all processes in phase $R+1$
    fi
    $[]R = t+1 \rightarrow decision_i := \text{NIL}$
  fi

$[]\, decision_i \neq \text{'?'} \rightarrow$ **halt**
fi

Figure 4.1 The early stopping algorithm

no process may decide on NIL while **m** is still a potential decision value for an undecided process.

Lemma 4.1

Consider the system of processes at the end of phase $R < (t+1)$. Let $P_i \notin FAULTY\text{-}RECEIVERS^R$ be a process which has not decided on **m** by this time. If there is a process P_j which first receives **m** in phase R then no process P_k has decided on NIL by the end of phase R.

Proof

The result follows from the size of $QUIET_k^R$ as determined by Lemma 3.1. □

It is now shown that the contrapositive of Lemma 4.1 is also true. That is, $|QUIET_k^R| < R$ is sufficient to conclude that **m** is no longer a viable decision value.

Lemma 4.2

Consider the system of processes at the end of phase $R < (t+1)$. Suppose there is a process P_k which has not decided on **m** such that $|QUIET_k^R| < R$. Then no non-faulty process may decide on **m** in phase $R' > R$. Furthermore, if $P_k \notin FAULTY\text{-}RECEIVERS^R$ then no non-faulty process ever decides on **m**.

Proof

Consider any non-faulty process P_i which has not reached a decision by the end of phase R. The protocol is such that P_i may decide on **m** in a subsequent phase only if there is some process P_j which first receives this value in phase R who will relay it to P_i. Suppose there were such a P_j. Then by Lemma 3.1 $|QUIET_k^R| \geq R$, a contradiction. Therefore there is no process able to relay **m** to any other process in later phases. Hence, no process (non-faulty or not) decides on **m** after phase R.

We now show that if $P_k \notin FAULTY\text{-}RECEIVERS^R$, then any process which has decided on **m** by the end of phase R must be faulty. By the preceding paragraph, any process P_j which has decided on **m** must have done so before phase R. But because $P_k \notin FAULTY\text{-}RECEIVERS^R$ has not decided on **m** by the end of phase R, P_j failed to relay its decision to P_k in some prior phase. Thus P_j is faulty.

It has therefore been shown that no non-faulty process can decide on **m** if $|QUIET_k^R| < R$ for $P_k \notin FAULTY\text{-}RECEIVERS^R$. □

The correctness of the algorithm of Figure 4.1 is now established.

Theorem 4.3

Consider a collection of n processes of which at most t commit send-faults. Let $n > (t + 1)$. The algorithm of Figure 4.1 correctly solves the Byzantine Generals Problem. Furthermore, if only $f < t$ send-faults occur, then all non-faulty processes have halted by the end of phase $(f + 2)$.

Proof

(Validity):

If *General* is non-faulty then all $P_i \notin FAULTY\text{-}RECEIVERS^1$ agree on **m** after one phase.

(Agreement):

Because a faulty process may not generate arbitrary values, it is clear that $decision_i \in \{0, 1\}$ implies $decision_i = \mathbf{m}$. Therefore should the output value of

any two non-faulty processes differ, it must be the case that one of the values is NIL. It is now demonstrated that if any non-faulty process P_i decides on **m** then all $P_j \notin FAULTY\text{-}RECEIVERS^{(t+1)}$ reach the same decision.

Suppose P_i decides on **m** at the end of phase R. If $R < (t+1)$ then P_i sends **m** to all processes in phase $(R+1)$. Therefore all processes which are undecided by the end of phase R and do not commit receive-faults with respect to P_i in phase $(R+1)$ will also decide on **m**. We show that no non-faulty process has decided on NIL by the end of R.

A non-faulty process may decide on NIL only if it received NIL from some other process or concluded on its own that **m** was not viable. Both these cases are shown to be impossible. We claim that no process could have sent the NIL value by the end of phase R. (Suppose otherwise. Then there is a process P_k such that $|QUIET_k^{R'}| < R'$ for $R' < R$. Because non-faulty process P_i first decides in phase R, P_i had not decided by the end of R'. By Lemma 4.2 P_i could not have decided on **m** in phase R, a contradiction.) Also no non-faulty process decides NIL on its own in phase R for this would violate Lemma 4.1. Hence no process has yet decided on NIL.

Therefore all non-faulty processes decide on **m** by the end of phase $(R+1)$.

For the case when $R = (t+1)$ we invoke Lemma 3.1 which tells us that if P_i has not decided on **m** by the end of phase t then there at least t processes in $FAULTY\text{-}RELAYERS^t$ (and hence also in $FAULTY\text{-}SENDERS^t$). Because P_i has

not received **m** from these processes they must all have committed send-faults. By assumption this is the maximum number of send-faults which may occur. Therefore whatever process P_k sent **m** to P_i in phase $(t+1)$ cannot commit a send fault. Hence all undecided $P_j \notin FAULTY\text{-}RECEIVERS^{(t+1)}$ receive **m** from P_k and reach the same decision as P_i.

We now prove that if $f < t$ send-faults actually occur, then all non-faulty processes have halted by the end of phase $(f+2)$.

It is clear from the structure of the protocol that as soon as the first non-faulty process reaches a decision then all non-faulty processes reach the same decision and halt by the end of the next phase.

Suppose P_i is the first non-faulty process to decide and that it does so at the end of phase R. If $decision_i = \mathbf{m}$ and there is still a non-faulty P_j which has not decided then $f \geq R$ by Lemma 3.1. Thus all non-faulty processes receive **m** from P_i by phase $(R+1) \leq (f+1)$.

On the other hand, suppose $decision_i$ = NIL. Because P_i was the first non-faulty process to decide, it must be the case that $|QUIET_i^R| < R$ but $|QUIET_i^{(R-1)}| \geq (R-1)$. Since P_i is non-faulty, only those processes which actually committed send-faults with respect to P_i are included in $QUIET_i^{(R-1)}$. Therefore $f \geq R-1$. All non-faulty process receive NIL from P_i and halt in phase $(R+1) \leq (f+2)$. □

5. Complexity

The complexity of our early-stopping protocol is now analyzed and compared to previous results. In each phase, a non-faulty process sends a fixed-length message to all others. Therefore, no more than $O(f \cdot n^2)$ message bits are used. This is exactly the complexity of the crash-fault protocol of [LF82]. Thus, lessening the failure restrictions does not entail additional cost. This contrasts with the protocol of [Hadz83] in which admitting a class of failure more restrictive than ours resulted in greater bit complexity.

6. Conclusions

A new model of process failure was presented in this paper. In our model, a faulty process may refuse to send or receive any message called for by the protocol; however it may not lie. This model subsumes the previously-treated modes of failure-by-halting and failure-by-omission.

We presented an early-stopping solution to the Byzantine Generals Problem for the new model. In doing so we demonstrated protocols which rendered receive-faults benign. Thus it was shown that only send-faults need be counted in process failures.

Our model is particularly appropriate for dealing with message system faults. Previous work assumed that a lost message could be handled by arbitrarily attributing the loss to a fault of one of the communicating processes. Doing so under-estimates resiliency; the immunity of our protocols to receive-faults is proof of this. Hence our model is more accurate in accounting for lost messages.

7. References

[Cris84]
Cristian, F., Aghili, A., Strong, R., and Dolev, D. *Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement.* Research Report RJ4540, I.B.M. Research, San Jose, CA, December 1984.

[DS83]
Dolev, D., and Strong, H.R. "Authenticated Algorithms for Byzantine Agreement". *SIAM Journal of Computing*, vol. 12, no. 4 (November 1983), pp. 656-665.

[FL82]
Fisher, M., and Lynch, N. "A Lower Bound for the Time to Assure Interactive Consistency" *Information Processing Letters*, vol. 14, no. 4 (1982), pp.183-186.

[Hadz83]
Hadzilacos, V. *Byzantine Agreement Under Restricted Types of Failures (Not Telling the Truth is Different from Telling Lies.* Computer Science Technical Report TR-18-63 , Harvard University, 1983.

[LF82]
Lamport, L., Fischer, M. *Byzantine Generals and Transaction Commit Protocols.* Opus 62, SRI International, April 1982.

[LSP82]
Lamport, L., Shostak, R., and Pease, M. "The Byzantine Generals Problem". *ACM Trans. on Programming Languages and Systems*, vol. 4, no. 3 (July 1982), 382-401.

[PSL80]
Pease, M., Shostak, R., and Lamport, L. "Reaching Agreement in the Presence of Faults". *Journal of the ACM*, vol. 27, no. 2 (1980), pp.228-234.

[Per84a]
Perry, K. *Early Stopping Protocols for Fault-Tolerant Distributed Agreement.* Ph.D. dissertation, Tech. Report TR 85-662, Dept. of Computer Science, Cornell University, February 1985.

[Per84b]
Perry, K. "Randomized Byzantine Agreement". *IEEE Trans. on Software Engineering*, vol. SE-11, no. 6 (1985), pp.539-546.

[Rabi83]
Rabin, M. "Randomized Byzantine Generals". *Proc. 24th Symp. on Foundations of Computer Science*, Tuscon, AZ, November 1983, pp.403-409.

[Toue84]
Toueg, S. "Randomized Byzantine Agreements". *Proc. Third Symp. on Principles of Distributed Computing*, Vancouver, BC, Canada, August 1984, pp.163-178.

[TPS85]
Toueg, S., Perry, K., and Srikanth, T.K. "Fast Distributed Agreement", *Proc. Fourth Symp. on Principles of Distributed Computing*, Minaki, Canada, August 1985.

[TC84]
Turpin, R. and Coan, B. "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement". *Information Processing Letter*, vol. 18, pp.73-76, February 1984.