Op. 62

Byzantine Generals
and
Transaction Commit Protocols

Leslie Lamport
Computer Science Laboratory
SRI International

Michael Fischer
Computer Science Department
Yale University

28 April 1982

minor revision: 25 April 1984

The first author's work was supported in part by the National Science Foundation under grant number MCS-8104459. The second author's work was supported in part by the National Science Foundation under grant number MCS-8116678 and by the Office of Naval Research under grant number N00014-82-K-0154.

ABSTRACT

The transaction commit problem in a distributed database system is an instance of the Weak Byzantine Generals problem. It is shown that even under the assumption that a process can fail only by "crashing"—failing to send any more messages—a solution to this problem that can tolerate k failures must, in the worst case, require at least $k + 1$ message-passing delays. Under this same assumption, a simple solution that exhibits the optimal worst-case behavior is given.

Contents

1 Introduction

In many database systems, there is a point in the processing of a transaction when an irrevocable decision is made whether to *abort* or *commit* it—where committing the transaction involves inserting any changes it made into the database. For a distributed database system, this decision must be announced to all the sites affected by the transaction. We will show that designing a fault-tolerant protocol for arriving at this decision is equivalent to the Weak Byzantine Generals (WBG) problem introduced in [7], which is closely related to the Byzantine Generals problem described in [6] and [8].

These Byzantine Generals problems have been studied under very weak assumptions about the behavior of a failed process—in particular, when a failed process can "lie" by sending contradictory information to two other processes. Solutions have been found that work despite these weak assumptions. Moreover, these solutions have been shown to be optimal in certain respects. In particular, it has been shown that a solution which handles up to k process failures must, in the worst case, require at least $k + 1$ message-passing delays [1, 4].

The transaction commit protocol has been studied under a much stronger assumption about process failure—namely, that a failed process simply "dies", and that its death is detectable by other processes. We will call such a simple failure a "crash". Processes that can fail only by crashing have been called *fail-stop* processes [9]. Since work on the Byzantine Generals problems has concentrated on handling lying processes, there seems to be a general feeling that this work is not relevant when processes fail only by crashing. The main result of this paper is that any solution to the WBG problem, and hence any transaction commit protocol, that can handle up to k crashes also requires at least $k + 1$ message-passing delays in the worst case. Thus, restricting failure to be simple crashes does not improve the worst-case time complexity of transaction commit. A weaker result of this kind was obtained by Dolev and Strong [3], who restricted processes to fail only by failing to send messages, but they allowed a failed process to continue sending messages after it fails. A result similar to ours but somewhat weaker was also obtained independently by V. Hadzilacos [5].

We also demonstrate a solution that can handle up to k crashes such that if only f crashes actually occur, then every process will be aware of the outcome after at most $f+1$ message-passing delays. (See [2] for a discussion of early stopping of Byzantine Generals solutions.)

2 The WBG Problem

We assume a system consisting of n processes, numbered 1 through n, each of which can communicate directly with every other process by sending messages. In the weakest form of the transaction commit problem, each process initially chooses whether the transaction should commit or abort and then communicates with all the other processes to decide upon a final action—either committing or aborting the transaction. The following two conditions must be satisfied.

TC1. If no process fails and all processes initially choose the same action, then that is the final action taken by all processes.

TC2. Any two processes that do not fail take the same final action.

One might wish to strengthen this condition—for example, by requiring that, in the absence of failures, the transaction abort if any process initially chooses to abort it. For our lower bound on the time complexity, it suffices to use this weaker problem. The algorithm we give in Section 4 can be used to obtain such a stronger condition.

The transaction commit protocol is usually stated in terms of a transaction *coordinator*—a single process that makes the initial decision of whether to abort or commit. Condition TC1 is then replaced by the following:

TC2′. If no process fails, then the action taken by every process is the one initially chosen by the transaction coordinator.

Note that TC1′ is stronger than TC1—a protocol satisfying TC1′ *a fortiori* satisfies TC1. By considering the weaker condition TC1, we obtain a stronger lower bound in Section 3. The algorithm in Section 4 satisfies a condition even stronger than TC1′.

In the WBG problem, the two actions *commit* and *abort* are generalized to arbitrary values in some set V. Each process initially chooses an initial "private" value, and then chooses a final "public" value after communicating with the other processes. TC1 and TC2 have the following obvious generalizations:

WBG1. If no process fails and all processes initially choose the same private value, then that value will be chosen by all processes as their public value.

WBG2. Any two processes that do not fail choose the same public value.

3 The Time Complexity of the WBG Problem

In this section, we prove our result on the worst-case time complexity of any solution to the WBG problem. Reasoning about Byzantine Generals problems tends to be rather subtle, and informal reasoning can easily lead to errors. We will therefore formally define all our concepts, and give a very detailed proof of our result. We use the following notation:

V: The set of all possible values that processes can choose.

M: The set of all possible messages that processes can send.

$|S|$**:** The cardinality of S, for any finite set S.

S^***:** The set of finite sequences of elements of S, for any set S. The empty sequence is denoted by Λ.

$s \preceq t$**:** If $s = s_1, \ldots, s_i$, then $t = s_1, \ldots, s_j$ for some $i \leq j$—i.e., s is an initial subsequence of t.

[k]: The set of integers $1, \ldots, k$, for any integer $k \geq 0$. Note that **[0]** is the empty set, which we also denote by ϕ.

[0,k]: The set of integers $0, \ldots, k$, for any integer $k \geq 0$. Note that **[0,0]** $= \{0\}$.

In any WBG algorithm, the processes first choose their initial private values, then send messages among themselves in order to agree upon a public value. We model the message-passing in terms of distinct rounds, where a process does not send any round r messages until it has received all round $r - 1$ messages that were sent to it. It is easiest to think about this model as describing a synchronous algorithm, where processes use some form of synchronization external to the algorithm (such as synchronized clocks) to decide when to begin the next round. However, the model can also represent an asynchronous algorithm, in which the sending of a message is caused only by the receipt of a message, or by a timeout that indicates the failure of a message to arrive. For such an asynchronous algorithm, we simply define a process's round r messages to begin with the first message whose contents depends upon either a round $r - 1$ message received from another process or a timeout indicating the absence of such a round $r - 1$ message.

We first define a *scenario* to be a single execution of such an algorithm, excluding the processes' actual choice of their public values. We consider scenarios in which a process can fail only by crashing—simply failing to send any further messages. Moreover, we assume that a process can detect the absence of a message, so it can tell if the sender crashed before sending the message.

Definition 1 *A k-round scenario $\mathcal{S}$ consists of the following:*

- *A value* $\mathrm{VAL}_p[\mathcal{S}] \in \mathbf{V}$*, for each* $p \in [\mathbf{n}]$*.*

  [$\mathrm{VAL}_p[\mathcal{S}]$ denotes the private value initially chosen by process p.]

- *A set* $\mathrm{CR}[\mathcal{S}](r)$*, for each* $r \in [\mathbf{k{+}1}]$*, such that* $\mathrm{CR}[\mathcal{S}](r) \subset \mathrm{CR}[\mathcal{S}](r+1)$ *for all* $r \in [\mathbf{k}]$*. We let* $\mathrm{C}[\mathcal{S}]$ *denote* $\mathrm{CR}[\mathcal{S}](k+1)$*.*

  [$\mathrm{CR}[\mathcal{S}](r)$ denotes the set of processes that have crashed before sending any round r messages. $\mathrm{C}[\mathcal{S}]$ is the set of all processes that crash sometime in the scenario.]

- *A sequence* $\mathrm{MSG}_p[\mathcal{S}](r) \in (\mathbf{M} \times [\mathbf{n}])^*$*, for all* $p \in [\mathbf{n}]$ *and* $r \in [\mathbf{k}]$*, such that* $\mathrm{MSG}_p[\mathcal{S}](r) = \Lambda$ *if and only if* $p \in \mathrm{CR}[\mathcal{S}](r)$*.*

  [$\mathrm{MSG}_p[\mathcal{S}](r) = (m_1, q_1), \ldots, (m_s, q_s)$ means that the i^{th} message sent by process p in round r is the message m_i having process q_i as its destination. Process p crashed before sending any messages in round r if and only if $\mathrm{MSG}_p[\mathcal{S}](r) = \Lambda$.]

Note that a process p can send several different messages to the same process during a single round. It can also send messages to itself. We are assuming that a process which has not crashed sends at least one message during each round. This assumption is made only to simplify our proof; without it, we would have to distinguish a process that crashes in round r after sending all its round r messages and one that crashes in round $r+1$ before sending any messages in that round. It is not a substantive restriction because any algorithm in which a process sends no messages during some round can be modified by having the process send a null message to itself.

We next define some relations between scenarios.

Definition 2 *For any k-round scenarios $\mathcal{S}$ and $\mathcal{T}$:*

- $\mathcal{S} =_r \mathcal{T}$*, for* $r \in [\mathbf{0{,}k}]$*, if*

  (i) $\mathrm{CR}[s](\mathcal{S}) = \mathrm{CR}[s](\mathcal{T})$ *for all* $s \in [\mathbf{r{+}1}]$

  (ii) $\mathrm{VAL}_p[\mathcal{S}] = \mathrm{VAL}_p[\mathcal{T}]$ *for all* $p \in [\mathbf{n}]$

(iii) $\mathrm{MSG}_p[\mathcal{S}](s) = \mathrm{MSG}_p[\mathcal{T}](s)$ *for all* $p \in [\mathbf{n}]$ *and all* $s \in [\mathbf{r}]$.

[$\mathcal{S} =_r \mathcal{T}$ means that the two scenarios are the same through round r.]

- $\mathcal{S} =_{r\backslash p} \mathcal{T}$, *for* $r \in [\mathbf{k}]$, *if:*

  (i) $\mathcal{S} =_{r-1} \mathcal{T}$

  (ii) $\mathrm{CR}[\mathcal{S}](r+1) \bigcup \{p\} \;=\; \mathrm{CR}[\mathcal{T}](r+1) \bigcup \{p\}$

  (iii) $\mathrm{MSG}_q[\mathcal{S}](r) = \mathrm{MSG}_q[\mathcal{T}](r)$ *for all* $q \neq p$.

  [$\mathcal{S} =_{r\backslash p} \mathcal{T}$ means that the two scenarios are the same through round r, except for the behavior of p during round r.]

- $\mathcal{S} \sim_{r|p} \mathcal{T}$, *for* $r \in [\mathbf{0{,}k}]$ *and* $p \in [\mathbf{n}]$, *if:*

  (i) $\mathrm{VAL}_p[\mathcal{S}] = \mathrm{VAL}_p[\mathcal{T}]$

  (ii) For all $u \in [\mathbf{r}]$ *and* $q \in [\mathbf{n}]$: $\mathrm{MSG}_q[\mathcal{S}](u)|_p \;=\; \mathrm{MSG}_q[\mathcal{T}](u)|_p$, *where* $(m_1, q_1), \ldots, (m_s, q_s)|_p$ *is the subsequence consisting of all the* (m_i, q_i) *such that* $q_i = p$.

  [$\mathcal{S} \sim_{r|p} \mathcal{T}$ means that the first r rounds of the two scenarios appear the same to process p.]

- $\mathcal{S} \sim \mathcal{T}$ *if there exists* $p \in [\mathbf{n}]$ *such that* $p \notin \mathrm{C}[\mathcal{S}] \bigcup \mathrm{C}[\mathcal{T}]$ *and* $\mathcal{S} \sim_{k|p} \mathcal{T}$.

  [$\mathcal{S} \sim \mathcal{T}$ means that the two scenarios appear the same to some process that does not fail in either of them.]

Note that the relations $=_r$, $=_{r\backslash p}$, $\sim_{r|p}$ and $\sim$ are all symmetric and reflexive.

We next define a *message-passing protocol* to be the set of possible scenarios generated by an algorithm. We assume that the protocol is deterministic—that is, the messages sent by a process during any round r depend only upon the initial private value of that process and the messages it receives through round $r-1$. There are two possible sources of nondeterminism for the message-passing protocol used by a WBG algorithm:

- The messages a process sends could depend upon the order in which messages sent to it are received, which might vary because of random transmission delays.

- A process could make a deliberate nondeterministic choice.

Since we are proving a nonexistence result, we can argue as follows that the nonexistence of an algorithm employing a deterministic protocol implies the nonexistence of one using a nondeterministic protocol. Given any nondeterministic protocol, we can remove the above two sources of nondeterminism to obtain a deterministic one by:

- Arbitrarily fixing transmission delays to eliminate any nondeterminism in message arrival time.
- Fixing in advance any choice made by a process.

If the algorithm works with the nondeterministic protocol, then it would still work when the nondeterminism is removed by these restrictions. Hence, it suffices to prove the nonexistence of an algorithm using a deterministic message-passing protocol.

This informal reasoning can be avoided by allowing nondeterministic protocols in our formal definition. However, there is another type of nondeterminism that can appear in a protocol which would make it very easy to define a WBG algorithm:

- Allowing the process's actions to depend upon the private value chosen by another process.

Of course, this kind of nondeterminism cannot be implemented in a real protocol, since there is no way for one process to know anything about the private values of other processes except through messages that it has received. It is rather difficult to devise a formal definition of a nondeterministic protocol that allows the first two kinds of nondeterminism but not the third. We have therefore chosen to avoid this problem and consider only deterministic protocols.

It would perhaps be most natural to define a protocol to be an algorithm (or formal machine) that generates scenarios. Instead, we formally define a protocol to be the set of all possible scenarios that can be generated. These two approaches are logically equivalent, since an algorithm can be identified with the set of behaviors that it can produce. We have chosen to view a protocol as a set of scenarios for the following reason. We want to place the weakest possible constraints on a protocol because this leads to the strongest possible lower-bound result—one that applies to the widest class of protocols. Weakening the constraints on a protocol means allowing a larger class of scenarios. However, specifying that an algorithm allow a larger class of behaviors usually requires the explicit addition of nondeterminism to the algorithm. If we were to define a protocol as an algorithm, trying to weaken the definition would require describing a more complex algorithm. On the other hand, with our approach, where we specify a protocol by placing conditions on the set of scenarios, allowing a larger class of behaviors means adding fewer conditions. This leads more naturally to the weakest possible definition.

Definition 3 *For any $k > 0$ and $c \geq 0$, a k-*round *c-*crash message-passing protocol $\mathbf{P}$ *consists of a set of k-round scenarios such that:*

1. *For all $\mathcal{S} \in \mathbf{P}$: $|\mathrm{C}[\mathcal{S}]| \leq c$.*

   [We restrict our attention to scenarios in which at most c processes crash.]

2. *For any values $v_1, \ldots, v_n \in \mathbf{V}$ there exists a scenario $\mathcal{S} \in \mathbf{P}$ such that:*

   (i) *For each $p \in [\mathbf{n}]$: $\mathrm{VAL}_p[\mathcal{S}] = v_p$.*
   (ii) $\mathrm{C}[\mathcal{S}] = \phi$.

   [The processes can choose any initial private values, and need not fail.]

3. *For any $\mathcal{S}, \mathcal{T} \in \mathbf{P}$ and any $r \in [\mathbf{k}]$: if $\mathcal{S} \sim_{r|p} \mathcal{T}$ and $p \notin \mathrm{CR}[\mathcal{S}](r+1)$, then $\mathrm{MSG}_p[\mathcal{T}](r) \preceq \mathrm{MSG}_p[\mathcal{S}](r)$.*

   [This means that the messages sent by a process p during round r are determined by what it sees through round $r-1$, except that p could crash during round r.]

4. *For any $\mathcal{S} \in \mathbf{P}$, any $p \in [\mathbf{n}]$, and any $r \in [\mathbf{k}]$ such that*

   $$\mathrm{MSG}_p[\mathcal{S}](r) = (m_1, q_1), \ldots, (m_s, q_s)$$

   with $s > 0$, there exists a scenario $\mathcal{T} \in \mathbf{P}$ such that:

   (i) $\mathcal{S} =_{r \backslash p} \mathcal{T}$
   (ii) $\mathrm{MSG}_p[\mathcal{T}](r) = (m_1, q_1), \ldots, (m_{s-1}, q_{s-1})$.

   [$\mathcal{T}$ is the scenario that is the same as $\mathcal{S}$ through round r, except that in $\mathcal{T}$, p crashes before sending the last round r message that it sent in $\mathcal{S}$.]

Note that in part 3 of this definition, if p is not in $\mathrm{CR}[\mathcal{T}](r+1)$ either, then interchanging $\mathcal{S}$ and $\mathcal{T}$ shows that $\mathrm{MSG}_p[\mathcal{S}](r)$ and $\mathrm{MSG}_p[\mathcal{T}](r)$ are equal, since they are both initial subsequences of the other. This implies that in part 4, p must be in $\mathrm{CR}[\mathcal{T}](r+1)$.

We now come to our formal definition of a WBG algorithm. In practice, a WBG algorithm must specify how any process chooses its public value as a function of its initial private value and the messages it has received. Rather than considering how this is done for every process, we define an *abstract WBG algorithm* to be a single mapping from scenarios to values which assigns to a scenario the public value chosen by every process that does not crash in this scenario. We can do this because Condition WBG2 implies that all processes which do not crash choose the same public value. This leads us to the following definition.

Definition 4 *A* k-round, c-crash abstract WBG algorithm *consists of a* k*-round, c-crash message-passing protocol* $\mathbf{P}$ *and a mapping* $w : \mathbf{P} \to \mathbf{V}$ *such that:*

1. *For any* $v \in \mathbf{V}$*, if* $\mathcal{S} \in \mathbf{P}$ *such that* $\mathrm{C}[\mathcal{S}] = \phi$ *and* $\mathrm{VAL}_p[\mathcal{S}] = v$ *for all* $p \in [\mathbf{n}]$*, then* $w(\mathcal{S}) = v$*.*

   [This is condition WBG1.]

2. *For any* $\mathcal{S}, \mathcal{T} \in \mathbf{P}$: *if* $\mathcal{S} \sim \mathcal{T}$ *then* $w(\mathcal{S}) = w(\mathcal{T})$*.*

   [If some process does not crash in either scenario and sees the same set of messages in both of them, then it (and hence, by WBG2, all noncrashed processes) choose the same public value in both.]

Our main result is the following.

Theorem I *If* $|\mathbf{V}| > 1$ *and* $n > k+1$*, then there does not exist a* k*-round,* k*-fault abstract WBG algorithm.*

The hypothesis $|\mathbf{V}| > 1$ is obvious, since there would be no problem if there were only one possible choice of public value. The hypothesis $n > k+1$ is needed because conditions WBG1 and WBG2 become essentially vacuous if $n-1$ processes crash. Thus, if $n = k+1$, then a k-round, $(k-1)$-fault algorithm is easily transformed into a k-fault algorithm.

To prove Theorem I, we need some lemmas and additional definitions. Part A of the following lemma states that the first r rounds of a scenario can always be continued to a complete scenario in which no processes crash after round r. Part B is similar, except that we can let one process fail at the beginning of round r.

Lemma 1 *For any* k*-round, c-crash message-passing protocol* $\mathbf{P}$ *and any scenario* $\mathcal{S} \in \mathbf{P}$:

A. For any $r \in [\mathbf{0,k}]$*, there exists a scenario* $\mathcal{T} \in \mathbf{P}$ *such that:*

(i) $\mathcal{T} =_r \mathcal{S}$

(ii) $\mathrm{C}[\mathcal{T}] = \mathrm{CR}[\mathcal{T}](r+1)$*.*

B. For any $r \in [\mathbf{k}]$ *and* $p \in [\mathbf{n}]$*, there exists a scenario* $\mathcal{T} \in \mathbf{P}$ *such that:*

(i) $\mathcal{T} =_{r\backslash p} \mathcal{S}$

(ii) $p \in \mathrm{CR}[\mathcal{T}](r)$*.*

(iii) $\mathrm{C}[\mathcal{T}] = \mathrm{CR}[\mathcal{T}](r+1)$*.*

Proof: The proof of part A is by induction on r. For $r = 0$, we let $\mathcal{T}$ be the scenario whose existence is guaranteed by part 2 of Definition 3, letting $v_p = \mathrm{VAL}_p[\mathcal{S}]$. Assume $r > 0$. Applying the induction hypothesis, we can find a scenario $\mathcal{T}' \in \mathbf{P}$ such that:

(i)$'$ $\mathcal{T}' =_{r-1} \mathcal{S}$

(ii)$'$ $\mathrm{C}[\mathcal{T}'] = \mathrm{CR}[\mathcal{T}'](r)$.

Let D denote the set of processes p such that $p \notin \mathrm{CR}[\mathcal{S}](r)$ and $p \in \mathrm{CR}[\mathcal{S}](r+1)$, so D is the set of processes that crash during round r of $\mathcal{S}$. For each $p \in D$, part 3 of Definition 3 implies that $\mathrm{MSG}_p[\mathcal{S}](r) \preceq \mathrm{MSG}_p[\mathcal{T}'](r)$. By successive application of part 4 of Definition 3, using part 3 to show that $=_{r-1}$ is maintained, we can construct a scenario $\mathcal{T} \in \mathbf{P}$ such that:

- $\mathcal{T} =_{r-1} \mathcal{S}$
- $\mathrm{C}[\mathcal{T}] = \mathrm{CR}[\mathcal{T}](r+1) = \mathrm{CR}[\mathcal{S}](r+1)$.
- $\mathrm{MSG}_p[\mathcal{T}](r) = \mathrm{MSG}_p[\mathcal{S}](r)$ for all $p \in D$.

If $p \notin D$, then either $p \in \mathrm{CR}[\mathcal{S}](r) = \mathrm{CR}[\mathcal{T}](r)$ or $p \notin \mathrm{CR}[\mathcal{S}](r+1) = \mathrm{CR}[\mathcal{T}](r+1)$. In both these cases, we have $\mathrm{MSG}_p[\mathcal{T}](r) = \mathrm{MSG}_p[\mathcal{S}](r)$—in the first case, since they both equal Λ by Definition 1, and in the second case by part 3 of Definition 3. Hence, $\mathrm{MSG}_p[\mathcal{T}](r) = \mathrm{MSG}_p[\mathcal{S}](r)$ for all p, which shows that $\mathcal{T} =_r \mathcal{S}$. This completes the proof of part A.

To prove part B, we first repeatedly apply part 4 of Definition 3 to get a scenario $\mathcal{T}' \in \mathbf{P}$ with $\mathcal{T}' =_{r\backslash p} \mathcal{S}$ and $\mathrm{MSG}_p[\mathcal{T}'](r) = \Lambda$, using part 3 to show that the $=_{r\backslash p}$ relation is maintained at each step. We then apply part A to $\mathcal{T}'$ to find the required scenario $\mathcal{T}$. ∎

Definition 5 *A k-round scenario $\mathcal{S}$ is* r-regular *if:*

(i) $|\mathrm{CR}[\mathcal{S}](s)| \leq s-1$ *for all* $s \in [\mathbf{r+1}]$

(ii) $\mathrm{C}[\mathcal{S}] = \mathrm{CR}[\mathcal{S}](r+1)$.

Thus, an r-regular scenario is one in which at most $s-1$ processes have crashed by the end of each round $s \leq r+1$, and no processes crash afterwards.

Lemma 2 *For any k-round message-passing protocol* $\mathbf{P}$*, any* $r \in [\mathbf{k}]$*, and any r-regular scenarios* $\mathcal{S}, \mathcal{T} \in \mathbf{P}$:

A. If $\mathcal{S} =_r \mathcal{T}$ *then* $\mathcal{S} = \mathcal{T}$.

B. If $\mathrm{C}[\mathcal{S}] = \mathrm{C}[\mathcal{T}]$, $\mathcal{S} \sim_{r|p} \mathcal{T}$ *for all* $p \notin \mathrm{C}[\mathcal{S}]$*, and* $|\mathrm{C}[\mathcal{S}]| < n$*, then* $\mathcal{S} \sim \mathcal{T}$.

Proof: The proof of part A is by backwards induction on r. For $r = k$, it is immediate from the definition. Assume $r < k$. Since $\mathcal{S} =_r \mathcal{T}$, $\mathrm{CR}[\mathcal{S}](r+1) = \mathrm{CR}[\mathcal{T}](r+1)$. The regularity of $\mathcal{S}$ and $\mathcal{T}$ then implies that $\mathrm{CR}[\mathcal{S}](r+2) = \mathrm{CR}[\mathcal{T}](r+2) = \mathrm{CR}[\mathcal{S}](r+1)$, so for each $p \in [\mathbf{n}]$ either:

1. $p \in \mathrm{CR}[\mathcal{S}](r+1) \bigcap \mathrm{CR}[\mathcal{T}](r+1)$, or
2. $p \notin \mathrm{CR}[\mathcal{S}](r+2) \bigcup \mathrm{CR}[\mathcal{T}](r+2)$.

In the first case, we have $\mathrm{MSG}_p[\mathcal{S}](r+1) = \mathrm{MSG}_p[\mathcal{T}](r+1) = \Lambda$. In the second case, part 3 of Definition 3 implies that $\mathrm{MSG}_p[\mathcal{S}](r+1) = \mathrm{MSG}_p[\mathcal{T}](r+1)$. This shows that $\mathcal{S} =_{r+1} \mathcal{T}$, and we can now apply the induction hypothesis to show that $\mathcal{S} = \mathcal{T}$, finishing the proof of part A.

To prove part B, we prove the stronger result that for any $r, s \in [\mathbf{k}]$ with $s \geq r$: if $\mathcal{S}$ and $\mathcal{T}$ are r-regular, $\mathcal{S} =_{r-1} \mathcal{T}$, $|\mathrm{C}[\mathcal{S}]| = |\mathrm{C}[\mathcal{T}]|$ and $\mathcal{S} \sim_{s|p} \mathcal{T}$ for all $p \notin \mathrm{C}[\mathcal{S}]$, then $\mathcal{S} \sim \mathcal{T}$. The proof is by backwards induction on s. For $s = k$, it follows immediately from the definition of $\sim$. Assume $s < k$. Since $r \leq s$, the assumption of r-regularity implies that $\mathrm{CR}[\mathcal{S}](s+2) = \mathrm{CR}[\mathcal{S}](s+2)$ and for all $p \in [\mathbf{k}]$, either:

(i) $p \notin \mathrm{CR}[\mathcal{S}](s+2)$, or

(ii) $p \in \mathrm{CR}[\mathcal{S}](s+1)$.

In the first case, we have $\mathrm{MSG}_p[\mathcal{S}](s+1) = \mathrm{MSG}_p[\mathcal{T}](s+1)$ by part 3 of Definition 3 (since each is a subsequence of the other), and in the second case we have $\mathrm{MSG}_p[\mathcal{S}](s+1) = \mathrm{MSG}_p[\mathcal{T}](s+1) = \Lambda$ by Definition 1. Hence, we can conclude that $\mathcal{S} \sim_{s+1|p} \mathcal{T}$ for all $p \in [\mathbf{k}]$, and the result now follows from the induction hypothesis. ∎

Definition 6 *The relation $\overset{*}{\sim}$ on $\mathbf{P}$ is defined to be the reflexive transitive closure of $\sim$, i.e., the smallest transitively closed relation such that*

(i) $\mathcal{S} \overset{*}{\sim} \mathcal{S}$

(ii) $\mathcal{S} \sim \mathcal{T}$ *implies* $\mathcal{S} \overset{*}{\sim} \mathcal{T}$.

The heart of our proof is the following result.

Lemma 3 *For any k-round, k-crash message-passing protocol $\mathbf{P}$ with $k < n-1$, any $p \in [\mathbf{n}]$, $r \in [\mathbf{k}]$ and r-regular scenarios $\mathcal{S}, \mathcal{T} \in \mathbf{P}$: if $\mathcal{S} =_{r\backslash p} \mathcal{T}$ and $p \in \mathrm{CR}[\mathcal{T}](r)$ then $\mathcal{S} \overset{*}{\sim} \mathcal{T}$.*

Proof: Let $\mathrm{MSG}_p[\mathcal{S}](r) = (m_1, q_1), \ldots, (m_s, q_s)$. The proof will be by double induction on r and s—forward induction on s and backwards induction on r. This involves proving the following cases:

1. $s = 0$, for any r;

2. $r = k$, assuming the lemma holds for $s-1$;

3. $r < k$, assuming the lemma holds for $r+1$ with any s, and for $r, s-1$.

For the first case, $s = 0$, we see that Definition 1 implies $\mathcal{S} =_r \mathcal{T}$, and the result follows from part A of Lemma 2.

We next consider the second case: $r = k$. Applying part 4 of Definition 3, we obtain a scenario $\mathcal{S}'$ such that $\mathcal{S}' =_{k\backslash p} \mathcal{S}$ and $\mathrm{MSG}_p[\mathcal{S}](r) = (m_1, q_1), \ldots, (m_{s-1}, q_{s-1})$. It is clear that $\mathcal{S}' \sim_{k|q} \mathcal{S}$ for all $q \neq q_s$. Since $|\mathrm{C}[\mathcal{S}]| < k < n-1$, there is at least one $q \notin \mathrm{C}[\mathcal{S}] \bigcup \{q_s, p\} = \mathrm{C}[\mathcal{S}'] \bigcup \{q_s\}$, so $\mathcal{S}' \sim \mathcal{S}$. Since $\mathcal{S}' =_{k\backslash p} \mathcal{T}$, we can apply the lemma for $s-1$ to conclude that $\mathcal{S}' \overset{*}{\sim} \mathcal{T}$, and the result now follows from the transitivity of $\overset{*}{\sim}$.

Finally, we consider the third case: $r < k$. We first apply part B of Lemma 1 to obtain a scenario $\mathcal{S}^{(1)} \in \mathbf{P}$ such that

- $\mathcal{S}^{(1)} =_{r+1\backslash p} \mathcal{S}$

- $\mathrm{C}[\mathcal{S}^{(1)}] = \mathrm{CR}[\mathcal{S}^{(1)}](r+2) = \mathrm{CR}[\mathcal{S}](r+2) \bigcup \{p\}$

This implies that $\mathcal{S}^{(1)}$ is $(r+1)$-regular. We also have $|\mathrm{C}[\mathcal{S}^{(1)}]| \leq r$, since $\mathrm{C}[\mathcal{S}^{(1)}] = \mathrm{C}[\mathcal{T}]$ and $\mathcal{T}$ is r-regular. We apply the lemma for $r+1$ to deduce that $\mathcal{S}^{(1)} \overset{*}{\sim} \mathcal{S}$.

We next apply part B of Lemma 1 to obtain a scenario $\mathcal{S}^{(2)} \in \mathbf{P}$ with:

- $\mathcal{S}^{(2)} =_{r+1 \backslash q_s} \mathcal{S}^{(1)}$
- $\mathrm{C}[\mathcal{S}^{(2)}] = \mathrm{CR}[\mathcal{S}^{(2)}](r+2) = \mathrm{CR}[\mathcal{S}^{(1)}](r+2) \bigcup \{q_s\}$.

Since $\mathcal{S}^{(1)}$ is $(r+1)$-regular, and $|\mathrm{C}[\mathcal{S}^{(1)}]| \leq r$, we see that $\mathcal{S}^{(2)}$ is $(r+1)$-regular. We can then apply the lemma for $r+1$ to conclude that $\mathcal{S}^{(2)} \overset{*}{\sim} \mathcal{S}^{(1)} \overset{*}{\sim} \mathcal{S}$, so $\mathcal{S}^{(2)} \overset{*}{\sim} \mathcal{S}$.

We next apply apply part 4 of Definition 3 to obtain a scenario $\mathcal{S}^{(3)} \in \mathbf{P}$ such that:

- $\mathcal{S}^{(3)} =_{r \backslash p} \mathcal{S}^{(2)}$
- $\mathrm{CR}[\mathcal{S}^{(3)}](r+1) = \mathrm{CR}[\mathcal{S}^{(2)}](r+1)$ (since $p \in \mathrm{CR}[\mathcal{S}^{(2)}]()$)
- $\mathrm{MSG}_p[\mathcal{S}^{(3)}](r) = (m_1, q_1), \ldots, (m_{s-1}, q_{s-1})$.

Note that the first r rounds of $\mathcal{S}^{(2)}$ and $\mathcal{S}^{(3)}$ are identical except that process p crashes one message sooner in $\mathcal{S}^{(3)}$ – before it sends its last message to q_s. However, q_s is in $\mathrm{CR}[\mathcal{S}^{(2)}](r+2) = \mathrm{CR}[\mathcal{S}^{(3)}](r+2)$. We can therefore apply part B of Lemma 2 to conclude that $\mathcal{S}^{(3)} \sim \mathcal{S}^{(2)}$. Since $\mathcal{S}^{(2)} \overset{*}{\sim} \mathcal{S}$, we have $\mathcal{S}^{(3)} \overset{*}{\sim} \mathcal{S}$.

Finally, since p sends only $s-1$ messages during round r of $\mathcal{S}^{(3)}$, we can apply the lemma for $r, s-1$ to conclude that $\mathcal{S}^{(3)} \overset{*}{\sim} \mathcal{T}$, which then gives the desired result $\mathcal{T} \overset{*}{\sim} \mathcal{S}$. ∎

We now prove Theorem I by assuming the existence of a k-round, k-fault WBG algorithm and deriving a contradiction. Let $\mathbf{P}$ and w be the protocol and mapping $w : \mathbf{P} \rightarrow \mathbf{V}$ comprising the abstract WBG algorithm. Let u and v be two distinct elements in V, and let $\mathcal{S}^{(j)}$ be the scenario in $\mathbf{P}$ such that $\mathrm{C}[\mathcal{S}^{(j)}] = \phi$ and

$$\mathrm{VAL}_p[\mathcal{S}^{(j)}] = \begin{cases} v & \text{if } p \leq j \\ u & \text{if } p > j \end{cases} .$$

The existence of the scenarios $\mathcal{S}^{(j)}$ is guaranteed by part 2 of Definition 3.

By part 1 of Definition 4, we have $w(\mathcal{S}^{(0)}) = u \neq v = w(\mathcal{S}^{(n)})$. Hence, there exists some $p \in [\mathbf{n}]$ such that $w(\mathcal{S}^{(p-1)}) \neq w(\mathcal{S}^{(p)})$. By part B of Lemma 1, there exists a 1-regular scenario $\mathcal{T} \in \mathbf{P}$ such that

- $\mathcal{T} =_{1\backslash j} \mathcal{S}^{(p-1)}$
- $\mathrm{C}[\mathcal{T}] = \{p\}$,

and a 1-regular scenario $\mathcal{T}' \in \mathbf{P}$ such that

- $\mathcal{T}' =_{1\backslash j} \mathcal{S}^{(p)}$
- $\mathrm{C}[\mathcal{T}'] = \{p\}$.

By Lemma 3, we have $\mathcal{T} \overset{*}{\sim} \mathcal{S}^{(p-1)}$ and $\mathcal{T}' \overset{*}{\sim} \mathcal{S}^{(p)}$. However, in $\mathcal{T}$ and $\mathcal{T}'$, all processes except p choose the same initial private value. Part 3 of Definition 3 implies that $\mathrm{MSG}_q[\mathcal{T}](1) = \mathrm{MSG}_q[\mathcal{T}'](1)$ for all $q \neq p$, and $\mathrm{MSG}_p[\mathcal{T}](1) = \mathrm{MSG}_q[\mathcal{T}'](1) = \Lambda$. We therefore have $\mathcal{T} \sim_{1|p} \mathcal{T}'$, so we can apply part B of Lemma 1 to conclude that $\mathcal{T} \sim \mathcal{T}'$, which implies $\mathcal{S}^{(p-1)} \overset{*}{\sim} \mathcal{S}^{(p)}$. However, it follows from part 2 of Definition 4 that this implies $w(\mathcal{S}^{(p-1)}) = w(\mathcal{S}^{(p)})$, which is the required contradiction. ∎

4 A Byzantine Generals Solution

In the ordinary Byzantine Generals Problem, there is a distinguished process, which we take to be process 1, with a private value. The goal is for every process to choose a value, which we call its "public" value, so the following two conditions are satisfied:

BG1. If process 1 does not fail, then every process that does not fail chooses process 1's private value as its public value.

BG2. Any two processes that do not fail choose the same public value.

Of course, BG2 is the same as WBG2. Condition BG1 generalizes TC1′ to an arbitrary set of values (instead of just *commit* and *abort*) and, more important, strengthens it to apply even when some processes fail. Given a solution to the Byzantine Generals problem, it is easy to construct a solution to the Weak Byzantine Generals problem, and hence to the transaction commit problem. However, the converse is not the case—the Byzantine Generals problem is in some sense harder than the Weak Byzantine Generals problem. We refer the reader to [6] for more details of the Byzantine Generals problem, and to [7] for a discussion of how it differs from the WBG problem.

In this section, we give a simple algorithm that solves the Byzantine Generals problem under the assumption that a process can fail only by crashing. It is a $(k+1)$-round algorithm that works in the presence of up to k crashes, which by Theorem I is the optimal worst-case time complexity. Moreover, we show that if only f processes actually do crash, then every process chooses its public value by the end of round $f+1$, and sends no messages after round $f+2$. The algorithm is described informally; we leave a formal definition of the message-passing protocol to the reader. Since process 1 could crash before sending any message, there has to be some default value in V chosen in that case. We let *null* denote that value.

Algorithm BG

Round 1: *Process 1 sends its value to every process.*

Round r, $1 < r \leq k+1$: *Each process does the following:*

1. *If it received a value from any process in round $r-1$, then it:*
   - *takes that value to be its public value;*
   - *sends that value to every process;*
   - *stops (executes no further round).*
2. *Otherwise, if every other process either:*
   (i) sent it an "I don't know" message in round $r-1$, or
   (ii) crashed before the beginning of round $r-1$ (as detected by the failure to receive any round $r-2$ message from that process)
   then it:
   - *takes* null *to be its public value;*
   - *sends the value* null *to every process;*
   - *stops.*
3. *Otherwise, it sends the message "I don't know" to every process.*

After Round $k+1$: *Each process (that has not stopped) does the following:*

1. *If it received a value from any process during round $k+1$, then it takes it as its public value.*
2. *Otherwise, it chooses* null *as its public value.*

Theorem II *Algorithm BG satisfies conditions BG1 and BG2 if at most k processes fail, and if a process can fail only by crashing. If at most f processes crash, then every process chooses its public value at or before the beginning of round $f+2$, and stops by the end of round $f+2$.*

Proof: If process 1 does not crash, then it successfully sends its value to every process during round 1, and every process that does not crash will then choose this value as its public value at the beginning of round 2. This proves condition BG1.

To prove BG2, we must show that if a process i that does not crash chooses a public value w, then any other process j that does not crash chooses the same public value. We may obviously assume that they do not both choose the value *null*, so we suppose that i chooses the value $w \neq null$. Since processes can fail only by crashing, no process sends the value w without receiving it from some other process—except for process 1, which can send its private value w in round 1. Hence, process j must either choose w or *null* as its public value. We show that it cannot choose the value *null*.

Assume that i chooses its public value w as the result of receiving a message sent during round r. During each round $s \leq r$, some process must have sent the value w to at least one other process. Hence, for each round $s \leq r$, there is at least one process that neither sent an "I don't know" message during round s nor crashed during a previous round. This implies that j could not have chosen the value *null* during any round up to or including round $r + 1$. If $r < k + 1$, then i must send the value w during round $r + 1$, and that value must be received by j because neither i nor j crash. Hence, in this case, j must choose the value w during round $r + 1$. If $r = k + 1$, then during each of the $k + 1$ rounds a different process sent the value w to all processes. By hypothesis, at most k processes crash. Hence, some process that did not crash sent the value w to all processes. Since j did not crash, it must have received that value and must have chosen it as its public value. This completes the proof of condition BG2.

Finally, assume that f processes crash, where $f < k$. A process chooses a value after any round in which it does not observe the failure of a process—i.e., during any round in which it receives a message from every process that it did not previously observe to have failed. Since there are at most f crashes, this must happen by the end of round $f + 1$. Once it chooses its value, a process stops after sending its next round of messages. ∎

If only $f < k$ processes crash, Algorithm BG lets every process chooses its public value by the beginning of round $f + 2$, which may also be considered the end of round $f + 1$. Theorem I implies that no algorithm can do better than this. However, one might hope to find any algorithm that avoids sending any round $f + 2$ messages. It can be shown that this is impossible. The proof is essentially the same as the proof of Theorem I, and will be omitted. Our algorithm is therefore optimal in terms of the number of rounds that it takes to stop.

References

[1] R. Demillo, N. Lynch and M. Merritt. Cryptographic Protocols. In *Proceedings of the 14th ACM SIGACT Symposium on Theory of Computing*, (May 1982).

[2] D. Dolev, R. Reischuk and R. Strong. "Eventual" is Ealier than "Immediate". *Proceedings of the 23rd Symposium on Foundations of Computer Science* (1982), 196-203.

[3] D. Dolev and R. Strong. Requirements for Agreement in Distributed Systems. *Proc. 2nd Symposium on Reliability in Distributed Software and Database Systems*, Pittsburgh (July 1982), 53-60.

[4] Michael Fischer and Nancy Lynch. A Lower Bound for the Time to Assure Interactive Consistency (1981).

[5] V. Hadzilacos. A Lower Bound for Byzantine Agreement with Fail-Stop Processors. Harvard University Center for Research in Computing Technology Technical Report TR-21-83 (July 1983).

[6] L. Lamport, R. Shostak and M. Pease. The Byzantine Generals Problem. *ACM Trans. on Prog. Lang. and Systems 4*, 3 (July 1982), 382-401.

[7] L. Lamport. The Weak Byzantine Generals Problem. *Journal of the ACM*

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

[9] F. B. Schneider. Byzantine Generals in Action: Implementing Fail-Stop Processors. *ACM Trans. on Computer Systems 2*, 2 (May 1984).