With less malicious Byzantine generals: Agreement algorithms under value faults

P.D. Ezhilchelvan

Abstract

Reaching agreement in the presence of faults is a fundamental problem in fault tolerant distributed computing. This paper describes, and establishes the correctness of, algorithms for reaching distributed agreement in the presence of faults which are less "malicious" than Byzantine faults. Consistent value faults and value faults are the two types of faults considered. Faults of these two types cause a processor to fail by producing incorrect output values. Assumptions have been made to exclude a faulty processor's "malicious" failure modes such as a faulty processor's ability to "impersonate" another faulty processor. However, it is considered to be possible for a processor to fail, due to "accidental" sources of errors, by "two-facing", i.e., by delivering one value to one processor and a different or no value to another processor. Taking an upper bound on the number of processors that can possibly fail, an agreement algorithm is developed for each type of fault. It has been shown that these algorithms are optimal with respect to time complexity. The development of these agreement algorithms illustrates that reaching agreement does not turn out to be a simple task, despite the exclusion of malicious failure modes of faulty processors.

Series Editor: M.J. Elphick

©1988 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

EZHILCHELVAN, Paul Devadoss.

With less malicious Byzantine generals: Agreement algorithms under value faults. [By] P.D. Ezhilchevan.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1988.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 252).

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.

Abstract

Reaching agreement in the presence of faults is a fundamental problem in fault tolerant distributed computing. This paper describes, and establishes the correctness of, algorithms for reaching distributed agreement in the presence of faults which are less "malicious" than Byzantine faults. Consistent value faults and value faults are the two types of faults considered. Faults of these two types cause a processor to fail by producing incorrect output values. Assumptions have been made to exclude a faulty processor's "malicious" failure modes such as a faulty processor's ability to "impersonate" another faulty processor. However, it is considered to be possible for a processor to fail, due to "accidental" sources of errors, by "two-facing", i.e., by delivering one value to one processor and a different or no value to another processor. Taking an upper bound on the number of processors that can possibly fail, an agreement algorithm is developed for each type of fault. It has been shown that these algorithms are optimal with respect to time complexity. The development of these agreement algorithms illustrates that reaching agreement does not turn out to be a simple task, despite the exclusion of malicious failure modes of faulty processors.

About the author

Mr. Ezhilchevan has been a Research Associate in the Computing Laboratory since September, 1983.

Suggested keywords

ALGORITHMS
DISTRIBUTED AGREEMENT
FAULT TOLERANCE
RESTRICTED FAILURES
VALUE FAULTS

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.64404 621,381958
U.D.C. 681.322.06/02
WITH LESS MALICIOUS BYZANTINE GENERALS: AGREEMENT ALGORITHMS UNDER VALUE FAULTS

Paul Devadoss Ezhichelvan
University of Newcastle upon Tyne, England, U.K.

1. INTRODUCTION

In this paper, we describe algorithms for reaching distributed agreement which are tolerant to faults of certain specific types. The problem of reaching distributed agreement, named by Lamport, Pease, and Shostak[Lampo82] as the Byzantine generals problem, can be described as follows: Consider a distributed system made up of n, n > 2, potentially faulty processors that are capable of communicating with each other only by message passing. It is assumed that the communication medium is fault free and that faults occur only in processors. The number of faulty processors, f, can be at most (n-2), i.e., f ≤ (n-1). A non-faulty processor can not, however, ascertain which other processors are faulty. Among these n processors in the system, one processor is designated as the sender and the others as receiver processors. The sender is to choose a value from many potential values and send it to all receiver processors so that they all receive the same value. When the sender is faulty, it cannot be guaranteed that all non-faulty receiver processors receive the same value directly from the sender. The problem is to develop an agreement algorithm that can be executed to guarantee that the following conditions will be met within some known, and bounded time interval:

C1 all non-faulty receiver processors decide on the same value, and
C2 when the sender is non-faulty, all non-faulty receiver processors decide on the value the sender chose to send.

In C1, it is ensured that all non-faulty receiver processors reach an unanimous decision on the sender's value; in C2, it is guaranteed that if the sender is non-faulty, then every non-faulty receiver processor reaches a valid decision by deciding on the value sent by the sender. When C1 and C2 are met, the agreement will be said to have been reached (on the sender's value) by non-faulty receiver processors. When the sender is non-faulty, it can be noted that C2 implies C1.

The development of agreement algorithms has been explored extensively under different types of faults. In [Pease80, Dolev83, Dolev82a], the type of faults considered for solving the agreement problem is taken to be the least restrictive one, known as Byzantine faults, in which no assumptions are made on responses produced by faulty processors. A faulty processor can exhibit malicious behaviour such as attempting to "impersonate" another processor, or colluding with other faulty ones in trying to "sabotage" any attempt on solving the agreement problem etc. Under this fault type, faster algorithms have been developed in [Baba085] using redundancy in communication medium. Omission and timing are the other two types of faults considered by Cristian et. al. in solving the agreement problem [Crist85]. In [Hadzia], an agreement algorithm has been developed under fail-
stop faults. Early stopping agreement algorithms have been developed in [Ezhi87, Lam834] considering fail-stop, omission, and timing faults. The types of faults considered here are: consistent value faults and value faults. Assumptions are made to exclude a faulty processor's "malicious" failure modes such as its ability to "impersonate" other faulty processors. However, it is considered to be possible for a processor to fail, due to "accidental" sources of errors, by "two-facing", i.e., by delivering one value to one processor and a different or no value to another processor. An agreement algorithm is developed for each type of fault and the algorithms are shown to be optimal with respect to time complexity. The algorithms are developed in the context mentioned above: in a distributed system of \( n \) processors that are capable of reliably communicating with each other by message passing, at most \( f \) processors can fail, while the receiver processors are to reach agreement on the sender's value which is to be broadcast by the sender at some unknown time. Modifying these algorithms to the general context of every processor in the system being a sender and sending a value to all other processors is straightforward.

The rest of this paper is organised as follows: In the next section, the fault types are described. The section 3 explains the assumptions involved in the design of the algorithms. The sections 4 and 5 respectively describe, and establish the correctness of, the consistent value fault tolerant algorithm and the value fault tolerant algorithm. In each of these two sections, time optimality of the algorithms is indicated.

2. DESCRIPTION OF VALUE FAULTS

A component's faults and failures are defined and classified with reference to its specification. A processor fails when its behaviour deviates from that specified. Failures are caused by faults. As a consequence of having received a service request from the environment, a correctly functioning processor will produce (i.e. respond with) an output value that is expected in accordance with its specification. The response of a processor for a given service request will be correct, if the output value is as expected and produced on time [Kope85]. The response of a faulty processor need not be as specified. The characterisation of faults in a processor is fairly well known [Schl83, Mohan83, Crist85, Ezhi86] and is briefly described below: A fail-stop fault causes a processor to stop functioning; the corresponding failure is called a fail-stop failure. A fault that causes a processor, for a given service request, not to produce a response is called an omission fault and the corresponding failure an omission failure. A timing fault causes a processor to produce the expected value for a given service request either early or late; the corresponding failure is a timing failure. A value fault causes a processor to respond, for a given service request, on time but with a wrong value. The corresponding failure is a value failure. A Byzantine fault is responsible for a Byzantine failure which is any violation from the specified behaviour. In particular, a Byzantine failure includes the possibility of a processor producing an arbitrary output for an input for which the processor, by specification, is not expected to respond by producing an output.

In this paper, it is our interest to be concerned with faults of value type. A processor with value faults does not produce an arbitrary output for an input that does not require an output to be produced; however, for an input requiring an output, it may fail by producing an output with incorrect value. Note that a processor's failure by "not producing a response" can be regarded to be
equivalent to "producing a null value on time" and hence value faults subsume omission faults. As Byzantine faults can cause failures of any type, value faults will form a proper subset of Byzantine faults. We proceed to consider value faults in the context where processors are required to produce replicated responses for a given input. For example, in Triple Modular Redundant systems [Lyons62] a processor is required to send its output to three other processors; similarly, when processors are taking part in some agreement protocol, every processor is required to send its output to every other processor in the system.

2.1. Value Faults and Consistent Value Faults

Suppose that a processor is required to produce a replicated output containing r individual outputs, where r is the specified replication level and r > 1, as a result of receiving an input that requires a replicated output to be produced. A non-faulty processor will produce a correct replicated output in which all the individual outputs will appear on time and with correct, hence identical, values. A value fault causes a processor to produce the individual outputs on time but with incorrect values. The corresponding failure will be called a value failure. Note that in a value failure the individual outputs produced need not have identical values. If, in a value failure, all individual outputs contain identical values, then the value failure will be called a consistent value failure. A consistent value failure will be said to be caused by a consistent value fault. (When all the individual outputs have null values, the consistent value failure will represent consistent omission failure.) Consistent value faults form a proper subset of value faults and the failures caused by them will represent "less serious" violation from that specified. (Like consistent value faults, derived can be consistent omission faults, consistent timing faults, etc. For a complete fault classification, we refer the interested readers to [Ezhi86].)

2.2. Malicious Failure Modes

In a (consistent) value failure, the output value will be incorrect and, by definition, it can be any random value other than the correct one. Due to this randomness of incorrect output values, a processor with (consistent) value faults can have "malicious" failure modes. To illustrate this point, consider a distributed system, made up of processors and links, in which processors communicate with each other only by exchanging messages over links. Suppose that, in an attempt to detect erroneous input messages, processors have secret encryption key to sign every message to be sent and a public decryption key to authenticate every signed message received [Rives78]. A component with Byzantine faults need not identify an erroneous (i.e., an unauthentic) input message and may consequently produce an output instead of having to identify and subsequently ignore the erroneous input message. Since it can produce arbitrary outputs, it can, in principle, reveal its secret encryption key to another Byzantine faulty processor and thus enable that processor to forge its signature [Lampo82]; in other words, it can fail in malicious manner. A processor with (consistent) value faults will not produce an output for an input that does not require any output to be produced. Therefore, the faulty processor will respond in the same manner as a non-faulty processor for an erroneous input that can be identified and should be subsequently ignored. In other words, the faulty processor will duly authenticate every signed message received. However, the faulty processor may fail by producing an output with incorrect value for an input that requires an output to be
produced. While responding to such inputs, the faulty processor can fail, by definition, by revealing its secret key to other processors and by attempting to forge another processor's signatures. This is because the incorrect output, in a value failure, can have any random value. Suppose that it can be assumed that a processor with (consistent) value faults will not fail with an intention of revealing its secret encryption key to other processors or with an intention of forging another processor's signature. With this assumption, the (consistent) value failures will represent "less malicious" or "accidental" failure modes such as responding with erroneously computed values due to residual software design faults, or unstable power supply etc. Approaches such as recovery blocks [Horni74, Lee78], and N-version programming [Schne86], are meant to provide tolerance for software design faults.

In this paper, we consider faults of types consistent value and value together with a set of assumptions that restrict, not totally eliminate, the failures caused thereof. Under these assumptions, a faulty processor will be restricted not to "deliberately" attempt to corrupt and relay another processor's signed message and not to "intentionally" attempt to forge another processor's signature. However, it may respond with erroneously computed values and may fail to relay what it has received. Due to its unstable power supply, its output signal may be so weak that one processor may interpret it as 0 and another as 1. (An accidental source of error leading to two-facing.) We believe that the assumptions made to restrict the randomness of (consistent) value failures are sufficiently realistic to model the real behaviour of (consistent) value faulty processors in distributed systems. In the next section, these assumptions, along with other assumptions necessary to develop agreement algorithms, are stated and explained.

3. ASSUMPTIONS

In this section, stated and explained are the system context chosen, and the assumptions made, for designing the agreement algorithms tolerant of consistent value faults and value faults.

In solving the agreement problem, we consider a context in which the sender's broadcast time is not known to receiver processors a priori. The agreement algorithms designed in this context will be directly applicable for building distributed facilities such as agreement and ordering abstractions [Schne86], atomic broadcast, A-common storage [Crist85], etc. In the context of sender's broadcast time not known a priori, the sender processor will be required to indicate its time of transmission in the form of a timestamp that is to be included in every message it broadcasts. So, the sender processor observes the following rule: if, at clock time $T_s$, a value is decided to be delivered to receiver processors, then the value will be broadcast with the timestamp $T_s$. The following assumption is made on the sender processor's timestamps.

Assumption A1:

The sender processor will not be necessitated to carry out more than one broadcast with the same timestamp.

Remark:

If, at the same clock time, two or more distinct values are decided to be delivered, then A1 will require the sender processor to transmit all these values in a single message with one timestamp.
Clock Synchronism

The physical hardware clocks of processors have small differences in their running rates and their readings tend to drift apart with the passage of real time. Design of any deterministic agreement algorithm will require the processors to observe time within some known and bounded difference. In order to meet the requirement, the processors have to periodically adjust the readings of physical clocks to counter the difference that has so far developed. However, the readings of a physical clock can not be directly adjusted and, therefore, every processor constructs, over its physical clock, a reference clock, called simply clock, whose readings can be altered. The readings of (reference) clocks are adjusted periodically through the execution of clock synchronisation algorithms [Halpe84, Crist86]. The following is assumed for clocks of non-faulty processors:

Assumption A2:

At any given instant of real time, the observable difference between clock readings of any two non-faulty processors will be at most $e$.

A processor with value faults does not fail by producing unexpected outputs. A faulty processor will identify an untimely message, where possible, and will ignore the message, if necessary. When a faulty processor does not have its clock in bounded synchronism, it may fail by producing an arbitrary output for an untimely input for which no output is expected to be produced. Therefore, to enable faulty processors to identify untimely messages, the assumption A2 is extended also to clocks of faulty processors.

Assumption A3:

The clock readings of any two processors at any given instant of real time will be observed to have been bounded by not more than $e$.

Remark:

Assumption A3 requires the following of a processor with value faults: any faulty processor should have a non-faulty physical clock; any execution of clock synchronisation algorithm should proceed without suffering any failures and clocks should be reset by the appropriate amount during every execution. Thus, the underlying assumptions in A4 are: the physical clock of any faulty processor is non-faulty and a faulty processor's processes that are responsible for maintaining bounded clock synchronism do not suffer any failures.

Message Signature and Authentication

The messages, on being transmitted between processors, may get corrupted due to faulty processors along the transmission path and may, thereby, have their contents altered. A corrupted message, on being received, can deliver a value other than what the source processor intended to deliver, if it is not detected to have been corrupted. In an attempt to avoid being misinformed by corrupted messages, each processor is assumed to have facilities to "sign" every message it sends, and to "authenticate" the signature of every message it receives in order to detect any apparent attempt to corrupt the message.

In [Rives78], provided are schemes for processors to generate message signatures such that the signatures are signer-dependent and contents-dependent, and to authenticate signed messages so
that any attempt to alter the contents will be detected. Through these schemes, it is also possible for processors to over sign an already signed message by considering the first signature as yet another piece of data, and to authenticate a multiply signed message by recursively authenticating every individual signature starting from the one that was last added on and ending with the first one. These schemes, when implemented, will guarantee the following:

(i) a processor's signature for a given message will be such that it is highly unlikely for any other processor to be able to generate the same signature for that message, and

(ii) any attempt to alter the contents of a signed message is highly likely to be detected by a processor that authenticates the signed message. In the following, stated are a few assumptions made on signature and authentication capabilities of processors.

**Assumption A4:**

A non-faulty processor's signature for a given message can not be generated by any other processor and any attempt to alter the contents of a message signed (or over signed) by a non-faulty processor can be detected.

**Remarks:**

From what is guaranteed by schemes for generating message signatures and for message authentication, it can be seen that there exists a non-zero probability of (i) a faulty processor being able to generate the same signature that a nonfaulty processor would generate for a given message, and (ii) the contents of a message signed by a non-faulty processor being altered in such a way that the corrupted message can not be detected as unauthentic. In A4, this probability is assumed to be zero. Such an assumption has often been made in the literature [Dolev83, Crist85, Lamp082] for designing (what are called signed message) agreement algorithms.

A faulty processor does not fail by producing an unexpected output for a given input. A faulty processor will identify an unauthentic message, when it can, and will ignore the message, if necessary. When a faulty processor is not able to detect the unauthenticity of a corrupted or forged message, it may fail by producing an arbitrary output by accepting an unauthentic message for which no output is to be produced. Hence what is assumed for non-faulty processors is also extended to faulty processors:

**Assumption A5:**

A faulty processor's signature can not be generated by any other processor and any attempt to alter the contents of a signed message can be detected.

**Remark:**

In A5, a faulty processor is expected (i) not to reveal its private key that should be kept secret for it to be able to generate unforgeable signatures, and (ii) not to alter the contents of a signed message with an intention of making the corrupted message appear authentic. In other words, any attempt by a faulty processor to forge signatures or to alter the contents of signed message is considered to be more "accidental" than "intentional" in nature. The underlying assumption in A5 is that the attempts of a processor with value faults to generate another processor's signature or to alter the contents of a signed message is so accidental in nature that the signature and
authentication mechanisms are effective enough for such attempts to be detected.

**Bounded Communication Delay**

The processors in the distributed system communicate only by message passing. The communication medium is assumed to be reliably connecting every processor to every other processor in the system. The next assumption bounds the message transmission delay.

**Assumption A6:**

If, at time \( T \), a processor \( p \) decides to send a message to another processor \( q \), then \( p \) can form, and send, a message such that the message can be received in \( q \) at time \( T_r \), \( T \leq T_r < T + d \) - where time is measured according to a non-faulty processor's clock.

**Remark 1:**

The assumptions underlying A6 are: processors are reliably connected; \( d \) is fixed considering all events whose occurrences will require a processor to decide on sending messages, message routing and message traffic in the communication medium, and processing loads in processors.

**Remark 2:**

In the context where a communication path between two processors may be made up of communication links and other processors, assumption A6 does not require processors to be connected only by fault free paths - communication paths made up of links and non-faulty processors. It requires at least one fault free path to exist between any two processors in the system. The messages arriving through faulty paths, if any, may get corrupted; due to assumptions A4 and A5, such messages can be identified.

**Remark 3:**

Based on A2 and A6, the following can be stated:

If an event occurs in a non-faulty processor \( p \) at time \( T \) according to \( p \)'s clock and causes a message to be formed and sent to another non-faulty processor \( q \), then the message will be received in \( q \) at time \( T_r \), \( T - e \leq T_r < T + d + e \), according to \( q \)'s clock.

**Remark 4:**

It should be noted that the assumptions A2 and A6 are essential for developing deterministic agreement algorithms; without them, a non-faulty processor can not decide whether a message has not been sent at all or is yet to be sent by another processor. With non-faulty processors not being able to solve this ambiguity, agreement algorithms can not be designed [Fisch83]. The assumption A1 is also essential due to the system context chosen to solve the agreement problem. The assumptions A3 and A5 are particular to faulty processors and exclude malicious failures from being considered for solving the problem.

**Unanimity and Validity Conditions**

In the context of sender's broadcast time not known \textit{a priori}, the sender processor is required to deliver its value with timestamps. Therefore, each decision made by a receiver processor will be associated with a timestamp. In this context, the conditions, C1 and C2, for agreement will be
modified as follows:

When the sender processor broadcasts its value with timestamp \( T_s \),

Unanimity:

all non-faulty receiver processors either reach the same decision for \( T_s \) by their clock time \( T_s + \Delta \), where \( \Delta, \Delta > 0 \), is known and bounded, or do not ever take any decision for \( T_s \), and

Validity:

when the sender is non-faulty, all non-faulty receiver processors decide for \( T_s \) by their clock time \( T_s + \Delta \) on the value sent by the sender.

By validity condition, it is ensured that all non-faulty receiver processors decide for \( T_s \) by their clock time \( T_s + \Delta \) on the non-faulty sender processor's value delivered with timestamp \( T_s \). The unanimity condition ensures unanimity in either of the two aspects: if non-faulty processors decide for a timestamp, they all do decide on the same value; otherwise, none of them ever makes any decision for that timestamp. The latter aspect arises due to the chosen context in which agreement is to be reached. In other words, if the receiver processors had known \emph{a priori} the time at which the sender processor is to deliver its value, then at the end of the execution of the agreement algorithm the non-faulty receiver processors could be instructed to make decision on some default value, if they could not do otherwise when the sender is faulty. So, in the assumed context, the condition C1 is modified to the unanimity condition mentioned above.

In the following sections, presented are the algorithms tolerant of a finite number, \( f, f \leq n-2 \), of distinct processors suffering faults of consistent value and value types. For each agreement algorithm designed, \( \Delta \) will be expressed as a function of \( d \) and \( e \), and the size of \( \Delta \) will indicate the fastness of the algorithm.

To help appreciate the fastness of these algorithms, we provide the size of \( \Delta \) for algorithms developed under other fault types and with the same set of assumptions except A3 and A5. We refer the interested readers to Crist85.

\[ \Delta \text{ for omission faults } = (f+1)d + e. \]

\[ \Delta \text{ for Byzantine faults } = (f+1)(d + e) \]

Note that the agreement problem becomes a non-problem under consistent omission fault type. Under this type of faults, a faulty sender fails by not sending any message to any receiver processor. So, based on A2 and A6, \( \Delta \) will be \( d + e \).

4. CONSISTENT VALUE FAULT TOLERANT ALGORITHM

Under consistent value fault assumptions, faulty processors, on producing a replicated output, send messages of identical contents in a timely manner. However, the messages may have incorrect value. Therefore, when the sender is faulty, it need not broadcast its messages with correct timestamp, and may carry out two or more distinct broadcasts with the same timestamp. Therefore, it is possible for a receiver processor to receive from a faulty sender two or more different values for the same timestamp. To detect such a state, every receiver processor, in its attempt to reach agreement on the sender's value for a given timestamp, maintains a value bag, denoted as V-bag, in which the
value and the timestamp of sender's messages will be stored. Another bag, called the time bag and denoted as T-bag, is also maintained to hold only the timestamps of sender's messages. The agreement algorithm tolerant of consistent value faults is described and presented below.

Algorithm

The sender processor is to send its value in a timestamped and signed message to all receiver processors. A receiver processor, on receiving a message directly from the sender, authenticates the message. If the message is found authentic and is found to have arrived at $T_r$ according to its clock such that $T_s - e \leq T_r < T_s + (d + e)$, where $T_s$ is the message timestamp, then the value and the timestamp in the message is stored - as a pair - in V-bag, the timestamp is stored in T-bag, and a copy of the sender's message is over signed and relayed to other receiver processors. If the sender's message is found either unauthentic or untimely or both, it is simply ignored.

When a receiver processor receives a doubly signed message (i.e. the sender's message signed and relayed by another receiver processor), it inspects the value and the timestamp of the message. If the pair is not present in V-bag and if the message is found authentic, an entry is made in V-bag and the timestamp, if not already in T-bag, is entered there.

Whenever the clock reads $T_s + \Delta$ with $\Delta = 2(d + e)$, for any entry $T_s$ in T-bag, if the V-bag contains only one value associated with $T_s$, then the decision for timestamp $T_s$ is made on that value; otherwise it is made on a default value. After deciding, the entries corresponding to $T_s$ are removed from V-bag and T-bag.

In the following presentation of the algorithm, the messages exchanged between processors are taken to be of record structure:

```plaintext
type M record
  v: value; T_r: Time; signatures: string of char
end;
```

Every message is signed and contains a value, $v$, and a timestamp, $T_r$. The sender processor executes the following algorithm.

sender:

```plaintext
var msg: M; local-value: value; clock: Time;
begin
  msg.v := local-value; msg.T_r := clock;
  sign-and-send(msg)
end.
```

The algorithm executed by a receiver processor accommodates two rounds of message exchange. The first round allows a receiver processor to receive the sender's message directly, and the second round through another receiver processor. The two-round algorithm is presented below:
const Δ = 2(d + e); maxm-rounds = 2;

var
    V-bag: set of (v1: value, T1: Time); T-bag: set of Time;
    msg: M; default, v2: value; clock, T\textsubscript{r}, T2: time; s: integer;

function TC: boolean;
    begin TC := (s > 1 or msg.T\textsubscript{s} − e ≤ T\textsubscript{r} < msg.T\textsubscript{s} + (d + e))
    end;

begin
    cobegin

1) receive(msg);
2) T\textsubscript{r} := clock;
3) s := no-of-signatures(msg);
4) if (msg.v, msg.T\textsubscript{s}) not in V-bag
5) then if authentic(msg) and TC

    then begin
6) store({msg.v, msg.T\textsubscript{s}}, V-bag);
7) if msg.T\textsubscript{s} not in T\textsubscript{-bag} then store(msg.T\textsubscript{s}, T\textsubscript{-bag});
8) if s < maxm-rounds
9) then sign-and-send(msg)

    end

coend;

cobegin

10) for any T2 in T\textsubscript{-bag}
11) if clock = T2 + Δ

    then begin
12) if {v2, T2} unique in V\textsubscript{-bag}
13) then decide(v2, T2)
14) else decide(default, T2);
15) V\textsubscript{-bag} := V\textsubscript{-bag}/T2; T\textsubscript{-bag} := T\textsubscript{-bag}/T2

    end

coend
end.

Explanation:

The constant, maxm-rounds, represents the maximum number of rounds of message exchange the algorithm should accommodate and is taken to be 2. The variable V\textsubscript{-bag} is defined to be a set of double element sets with one element being a value and another a timestamp. The T\textsubscript{-bag} is a set of timestamps. The boolean function, TC, contains the timeliness condition a message received (line 1) at time $T\textsubscript{r}$ (line 2) with s distinct processor signatures (line 3) should satisfy for being accepted. If a
received message satisfies the timeliness condition, TC becomes true; false, otherwise. In line 4, a message received is checked whether its value and timestamp are already in V-bag. If not so, in line 5, it is verified for its authenticity and timeliness. On being found authentic and timely, appropriate entries are made in V-bag and T-bag and if the message is not doubly signed (line 8), it is oversigned and sent (line 9).

Lines 10 to 15 represent the second part of the algorithm to be executed concurrently with the first part (lines 1 to 9). For any entry, T2, in T-bag, at the time the clock reads T2 + Δ (lines 10, 11), a decision is taken for T2 in lines 12 and 13. With T2 as timestamp, if \( \{v2, T2\} \) is the only entry in V-bag, a decision is taken on v2 for timestamp T2; if there are many entries in V-bag, a default decision is made. In line 15, entries corresponding to T2 are removed from V-bag and T-bag.

In the following, the two-round algorithm presented and explained above is shown to be correct. It is assumed that the statements in the algorithm can be executed in no time (this will require an increase on the value of d).

4.1. Correctness of The Algorithm

Theorem 1:

Any execution of the algorithm meets the unanimity and the validity conditions for \( \Delta = 2(d+e) \), in the presence of \( f, f < n-1 \), distinct processors suffering consistent value faults.

Proof:
Case 1: Non-faulty Sender

By assumptions A2 and A6, every receiver processor can find the sender's message authentic and to have arrived during the appropriate clock interval. By assumption A4, a faulty receiver processor can not forge sender's signature and its attempt to alter the contents of the sender's signed message will be detected by a non-faulty receiver processor that receives that message. So, if the sender had sent a value v2 with timestamp T2, then, while line 12 of the algorithm is being executed, every non-faulty receiver processor's V-bag will do contain an element \( \{v2, T2\} \) due to assumptions A2 and A6, and, by assumption A4, \( \{v2, T2\} \) will be the only element with T2 as the second entry. Thus, by assumptions A2, A4, and A6 the validity condition is met in any execution of the algorithm.

Case 2: Faulty Sender

When a receiver processor stores a value V and a timestamp Tₜ, as a pair, in its V-bag, we will say that the processor "obtains" a \( \{V, Tₜ\} \). Under value fault assumptions, no faulty processor accepts an untimely message or suffers any undue delay in relaying a message to other processors. Therefore, no receiver processor can obtain a \( \{V, Tₜ\} \) at or after its clock time \( Tₜ + 2(d+e) \). Let p and q be two non-faulty processors. The unanimity condition is shown to be met by showing that if p obtains \( \{V, Tₜ\} \) before its clock reads \( Tₜ + Δ \), then q will also obtain \( \{V, Tₜ\} \) before its clock time \( Tₜ + Δ \).

Let p obtain \( \{V, Tₜ\} \) through a message received directly from the sender and let q not do so. The processor p must have received the sender's message at its clock time \( Tᵢ \), \( Tᵢ < Tₜ + (d+e) \).
When it over-signs the sender’s message and relays the doubly signed message to q, q will receive p’s message before its clock reads $T_s + 2(d+e)$.

Suppose that p obtain $\{V,T_s\}$ by receiving and accepting a message sent by another receiver processor, say, r. If r is non-faulty, then it is not possible for q not to have obtained what p has obtained. Suppose that r is faulty. Due to consistent fault model, the doubly signed messages transmitted by r will be of identical contents. Since p has accepted r’s message, q would also find r’s message authentic and to be containing value V and timestamp $T_s$. By assumption A5, a faulty processor can not undetectably alter the contents of a signed message; hence, r can not undetectably alter the sender’s value and timestamp. Therefore, when the doubly signed message sent by r is found authentic, it will contain the sender’s value and timestamp which are respectively V and $T_s$.

A faulty processor, under value fault model, accepts a message from the sender, only when the message is found authentic and is received within the acceptable time window specified in function TC (here, when r’s clock reads less than $T_s + (d+e)$). Since r does not suffer any undue delay in transmitting its signed messages, by assumptions A3 and A6, q will receive r’s message before its clock reads $T_s + 2(d+e)$. Hence, q obtains $\{V,T_s\}$ before its clock time $T_s + 2(d+e)$.

Therefore, for any entry $T_s$ in T-bag, all non-faulty processors will decide on the same value at clock time $T_s + 2(d+e)$. Hence the unanimity condition, and, hence, the theorem.

4.2. Time Optimality

Though a faulty sender broadcasts messages of identical contents, the timestamp of these messages need not be correct. When the timestamp is incorrect, it is possible that one non-faulty receiver processor finds the message timely and another non-faulty receiver processor finds the message untimely. Therefore, the algorithm is designed to accommodate two rounds of message exchange between processors. The next theorem establishes the optimality of the algorithm with respect to time complexity.

Theorem 2

In the context of sender’s broadcast time not known to receiver processors a priori, there exists no one-round agreement algorithm tolerant of a faulty sender processor that suffers consistent value faults.

Proof:

We prove this theorem by contradiction. Suppose that there exists an one-round algorithm and that the unanimity condition is guaranteed to be met in every execution for some finite $\Delta$, $\Delta > 0$. Under this one-round algorithm, there must exist a round wherein the sender broadcasts its message in a timestamped (and/or signed) message. In an execution of the algorithm, suppose that the faulty sender processor, accidentally and incorrectly, inserts a timestamp $T_s'$ in stead of $T_s$, where $T_s' = T_s - \Delta$, and $T_s$ is the clock reading when the value was decided to be delivered (and, hence, is the correct timestamp). The sender’s message gets transmitted to receiver processors without any undue delay and takes zero to less than d units of clock time to be received by receiver processors. All processors in the system have their clocks synchronised within a difference of at most $e$ (assumption
A3). Let p and q be two non-faulty receiver processors and, with respect to sender’s clock, let p’s clock be slow and q’s clock be fast. Let p receive the sender’s message just before its (slow) clock is to read $T^c_s + \Delta$, while q receives the sender’s message after its (fast) clock has read $T^c_s + \Delta$. So, in this execution, whatever be the non-zero value of $\Delta$, the unaminty condition will not be met so long as $e$ remains a non-zero quantity. In [Dolev84], it has been established that $e$ would be at least as large as $d/2$. Since $d$ can not be zero, $e$ can not be guaranteed to be zero. Hence, the theorem.

5. VALUE FAULT TOLERANT ALGORITHM

Value faults subsume consistent value faults. By appropriately modifying the two-round algorithm tolerant of consistent value faults, the value fault tolerant algorithm is developed. Faulty processors, under value fault model, need not send messages of identical contents to other processors, while producing a replicated output. Therefore, the algorithm should allow for $(f+1)$ rounds of message exchange so that the unaminty condition is guaranteed to be met, when the sender processor is faulty. Furthermore, under this fault model and due to A5, faulty receiver processors can not undetectably alter the contents of a signed message (hence can not undetectably change the sender’s timestamp), do not accept an unauthentic message, and do not suffer any undue delay in sending messages. Therefore, the receiver processors’ authentic messages need not be checked for timeliness and the timeliness condition, TC, of the consistent value fault tolerant algorithm can be retained. A receiver processor may receive the sender’s message with timestamp $T^c_s$, just before its (slow) clock reads $T^c_s + (d+e)$. Allowing for further $f$ rounds of message exchange, and for the clock difference, $\Delta$ will be $(d+e) + fd + e$, i.e., $(f+1)d + 2e$.

To summarise, the value fault tolerant algorithm is developed from the algorithm tolerant of consistent value faults by modifying "maxm-rounds" to $(f+1)$ and $\Delta$ to $(f+1)d + 2e$. The algorithm executed by receiver processors is given below:

```
const $\Delta = (f+1)d + 2e$; maxm-rounds = $(f+1)$;

var
quad V-bag: set of {v1: value, T1: Time}; T-bag: set of Time;
msg: M; default, v2: value; clock, Tr, T2: Time; s: integer;
function TC: boolean;
begin TC := (s>1 or msg.Ts - e \leq Tr < msg.Ts + (d+e)) end;
begin
  cobegin
  1) receive(msg);
  2) Tr := clock;
  3) s := no-of-signatures(msg);
```
4) if (msg.\text{v}, msg.T_s) not in V-bag
5) then if authentic(msg) and TC
       then begin
        6) store((msg.\text{v}, msg.T_s), V-bag);
        7) if msg.T_s not in T-bag then store(msg.T_s, T-bag);
        8) if s < maxm-rounds
        9) then sign-and-send(msg)
       end
    cobegin
10) for any T2 in T-bag
11) if clock = T2 + \Delta
       then begin
12) if \{v2, T2\} unique in V-bag
13) then decide(v2, T2)
14) else decide(default, T2);
15) V-bag := V-bag/T2; T-bag := T-bag/T2
    end
end.

5.1. Correctness Of The Algorithm

Theorem 3

Any execution of the above algorithm meets the unanimity and validity conditions for 
\(\Delta = (f+1)d + 2e\) in the presence of at most \(f\), \(f<(n-1)\), distinct processors suffering value faults.

Proof:

When the sender processor is non-faulty, the theorem is true by theorem 1. Suppose that the 
sender is faulty. As in theorem 1, we will say that a processor "obtains" \(\{V, T_s\}\) when it stores \(\{V, \ T_s\}\) in its V-bag. Under value fault model, a faulty receiver processor (i) will not accept a message 
from the sender with timestamp \(T_s\) at or after its clock time \(T_s + (d+e)\), (ii) can not undetectably 
change the sender's timestamp (due to A5) and, (iii) will not suffer undue delay while relaying 
the sender's value. Therefore, when a receiver processor accepts an authentic message with timestamp 
\(T_s\) and s, \(1 \leq s\), distinct processor signatures, the clock of every processor in the system will read 
less than \(T_s + sd + 2e\) (due to A3). Furthermore, under this fault model, faulty processors do not 
produce an output for an input that does not require any output to be produced. Therefore, no faulty 
processor would output messages for an input message that contains \((f+1)\) distinct processor signatures 
or for a message with \(f\) or less signatures one of which being its own (in which case the message 
will either contain a \(\{V, T_s\}\) already present in V-bag or not be authentic). So, no processor 
would ever receive an authentic message with more than \((f+1)\) signatures. Therefore, it is not
possible for a non-faulty processor to obtain \{V, T_s\} at or after its clock time \(T_s + (f+1)d + 2e\). Consider two non-faulty processors p and q. The unanimity condition is shown to be met by showing that for any \{V, T_s\} when p obtains \{V, T_s\} before its clock time \(T_s + \Delta\), q will also obtain \{V, T_s\} before its clock time \(T_s + \Delta\).

Suppose that p obtains a \{V, T_s\} by receiving a message with \(s, 1 \leq s \leq f+1\), processor signatures and that q does not receive the message. Processor p must have received the message when q’s clock read less than \(T_s + sd + 2e\). If \(s \leq f\), since p is non-faulty, q will obtain \{V, T_s\} through the message signed and relayed by p before its clock time \(T_s + (s+1)d + 2e \leq T_s + \Delta\). If \(s = (f+1)\), since there can be at most f faulty processors, it is not possible for q not to obtain \{V, T_s\} before its clock time \(T_s + \Delta\). Hence the unanimity condition.

**Time Optimality**

In the scenario of a faulty processor failing to deliver its information to all other processors, it has been established that an algorithm that is guaranteed to terminate always in less than \((f+1)\) rounds can not be designed [Dolev82b]. Thus, the above algorithm is optimal with respect to the size of \(\Delta\) which is \((f+1)d + 2e\).

6. CONCLUSIONS

Algorithms for reaching distributed agreement in the presence of consistent value faults and value faults have been presented. It has been shown that the algorithm tolerant of consistent value faults should accommodate two rounds of message exchange between processors. Though the sender processor with consistent value faults broadcasts messages of identical contents, the timestamp of these messages need not be correct. When the timestamp is incorrect, some non-faulty receiver processors may accept the message while other non-faulty processors may find the message arriving untimely and subsequently ignore the message. The value fault tolerant algorithm accommodates \((f+1)\) rounds of message exchange between processors and the size of \(\Delta\) is larger than that for omission faults by \(e\), the bound on clock difference, and is smaller than the size of \(\Delta\) for Byzantine faults by \((f+1)e\). In developing these algorithms, it has been assumed that a faulty processor will not be able to forge another processor’s signature, will not be able to undetectably corrupt a signed message, and will not relay an unauthentic and/or untimely message. We believe these assumptions can be quite realistic in practical systems and the algorithms presented here make a significant contribution to the area of algorithms for reaching agreement in distributed systems.

7. ACKNOWLEDGEMENTS

This work was supported by the U.K. Science and Engineering Research Council. My thanks are due to Professor S. K. Shrivastava who commented on this paper.
References

Babao85.

Crist85.

Crist86.

Dolev82.

Dolev82a.

Dolev83.

Dolev84.

Ezhil86.

Ezhil87.

Fisch83.

Hadjia.

Halpe84.

Horn174.
Kopet85.

Lamp082.

Lamp84.

Lee78.

Lyons62.

Mohan83.

Pease80.

Rives78.

Schli83.

Schne86.

Schne88.