From *Proceedings of the Sixteenth IEEE Symposium on Reliable and Distributed Systems (SRDS-97)*, Durham, NC, October 22-24, 1997.

PROBABILISTIC VERIFICATION OF A SYNCHRONOUS ROUND-BASED CONSENSUS PROTOCOL*

Harpreet S. Duggal, Michel Cukier and William H. Sanders

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL 61801, USA
(duggal,cukier,whs)@crhc.uiuc.edu
http://www.crhc.uiuc.edu/PERFORM

Abstract

Consensus protocols are used in a variety of reliable distributed systems, including both safety- and business-critical applications. The correctness of a consensus protocol is usually shown by making assumptions about the environment in which it executes, and then proving properties about the protocol. But proofs about a protocol's behavior are only as good as the assumptions which were made to obtain them, and violation of these assumptions can lead to unpredicted and serious consequences. In this paper, we present a new approach for the probabilistic verification of synchronous round-based consensus protocols. In doing so, we make stochastic assumptions about the environment in which a protocol operates, and derive probabilities of proper and non-proper behavior. We thus can account for the violation of assumptions made in traditional proof techniques. To obtain the desired probabilities, the approach enumerates possible states that can be reached during an execution of the protocol, and computes the probability of achieving the desired properties for a given fault and network environment. We illustrate the use of this approach via the evaluation of a simple consensus protocol operating under a realistic environment which includes performance, omission, and crash failures.

1 Introduction

The problem of reaching consensus [1] among a group of processes[1] in the presence of faults [2] is a central problem in the area of distributed systems. The importance of the consensus problem stems from its widespread occurrence in the construction of practical reliable distributed systems, and the fact that it subsumes many other frequently occurring problems, such as the membership problem [3]. Solutions to the consensus problem take different forms, depending on the assumptions made about how a protocol executes (e.g., in a synchronous or asynchronous fashion) and about the faults that might occur during its execution. There are many good surveys on this problem, such as [4, 5].

Consensus protocols have largely been verified using formal methods. In doing so, analysts make assumptions about the type and number of faults that might be tolerated, and prove properties about a protocol, provided that these assumptions hold (e.g., [6, 7]). But the proofs obtained are only as good as the assumptions made in proving them, which, in practice, might be violated. Likewise, a protocol can often exhibit correct behavior even when the assumptions used in its proof are violated, therefore making the characterization of when it will operate correctly, overly conservative. For example, Gong et al. [8] have enumerated possible sequences of events that can occur during the execution of several consensus protocols, and show that correct behavior often occurs even when the number of faults known to be tolerated is exceeded. Probabilistic verification (e.g., [9] and [10], where the authors call it "probabilistic validation") seeks to assign probabilities to the occurrence of important properties, either by partially exploring the space necessary to obtain a full proof or by making more reasonable (stochastic) assumptions about the fault environment in which a protocol operates.

Probabilistic verification can be done in two ways: by performing a stochastic evaluation (e.g., [11]), or by coupling a logical proof with an evaluation of the coverage of the assumptions [12, 13] made in the proof. The later approach, if taken, results in a simpler stochastic analysis (limited to evaluation of the probability of occurrence of the assumptions), but is inherently (and sometimes overly) conservative, since as argued earlier, a protocol can function correctly even if the assumptions that were made in its proof are violated. On the other hand, a direct stochastic evaluation can be more accurate, since it can evaluate directly the probability of occurrence of a property even when assumptions made in a proof are violated. However, it is often more difficult to carry out, since the evaluation must encompass the protocol itself, as well as the assumptions.

* This work was supported, in part, by DARPA contract F30602-96-C-0315.

[1] We use the general term "process" to refer to an abstract entity that, in turn, can represent a physical processor or site, a software process or server, a sub-network of a larger communication network, etc.

In this paper, we present a new approach for probabilistically verifying round-based consensus algorithms that accounts for performance, omission, and crash failures. The approach enumerates possible states that can be reached during an execution of the protocol, and computes the probability of certain properties for a given fault and network environment. The efficiency of the approach comes from its exploitation of the fact that the rounds in such protocols proceed at regular, fixed intervals. The approach is illustrated by the probabilistic verification of a simple variant of the Byzantine Generals (BG) protocol [14]. The BG protocol is a simple synchronous consensus protocol which has been studied under various types of process and link failures. The variant we consider was designed to tolerate crash failures, and has been proven to operate correctly as long as less than a certain number of failures occurred. We evaluate it under the more realistic case (for certain environments) of performance, omission, and crash failures.

The remainder of the paper is organized as follows. Section 2 presents a description of the BG problem, and an algorithm to solve it under the assumptions of only process crashes. Section 3 presents a model that can be used for the probabilistic verification of the BG protocol under the assumed stochastic environment. Section 4 presents the results of executing our model. Section 5 concludes the paper, suggesting how the approach we have taken can be applied to other round-based protocols.

2 Protocol Description

The BG problem [14, 4, 15] is a widely studied consensus problem. Solutions to it aim to achieve consensus in a system of n independent processes. Among these n processes, there is a distinguished process, called the *general*, possessing a *private value* before the execution of the protocol. The remaining n-1 processes are called the *lieutenants*. The goal of the protocol is for every lieutenant to choose a *public value*, such that the following two conditions are satisfied [15].

BG 1: If the general does not fail, then every process that does not fail chooses the general's private value as its public value.

BG 2: Any two processes that do not fail choose the same public value.

Informally, the above conditions collectively require that all processes that have not failed choose a unanimous public value, which is equal to the general's value if it has not failed.

Solutions to the BG problem depend on the assumptions made about the model of computation and the kinds of faults that are considered. In this section, we review an algorithm for solving the BG problem under the assumption that processes can fail only by *crashing* [16, 17], first presented in [15]. The protocol, as designed, assumes the availability of a fully "reliable" and "synchronous" message transmission mechanism. In this context, *reliable* implies that transmitted messages are always delivered to their intended recipients, and *synchronous* means that the time it takes a process to send a message is upper-bounded by a known deterministic value. Together, the assumptions about reliability and synchrony ensure that messages may never get lost or delayed due to the limitations of the network. The algorithm does, however, assume that processes may crash during the act of transmitting messages, thereby causing messages not to be delivered to all the intended recipients [2]. Figure 1 provides an algorithm that solves the BG problem under these assumptions.

Round 1: General sends his value to all his lieutenants.
Round r, $1 < r \leq K$: Each process does the following:
If it received a value from any other process in round r-1.
Then it:
a) takes it to be its public value;
b) sends that value to every other process;
c) stops (executes no further round).
Else 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:
a) takes null to be its public value;
b) sends null to every other process;
c) stops (executes no further round).
Else it sends an "*I don't know*" to every other process.
After Round K**:** Each process that has not stopped does the following:
If it received a value from any process in round K, then it takes it to be its public value;
Else it chooses null as its public value.

Figure 1: Classical BG Algorithm for Crash Failures.

The algorithm is a K=m+1 round protocol that has been proven to work in the presence of up to m=n-3 process crashes [15, 4] (m is called the *resiliency* of the protocol). Since the general could crash before sending any messages, there must be some pre-defined value which the lieutenant processes may agree on. This value is referred to as the *null* value, whereas the private value of the general is commonly referred to as the *valid* value.

Informally, the algorithm functions as follows. Execution begins with the general broadcasting his private value to all his lieutenants. Even though a perfectly reliable, synchronous communication network is assumed, some of the lieutenant processes might not receive the private value, since the general could have crashed in the midst of performing his broadcast. At the end of the first round, all lieutenants fall into one of three categories: those that have crashed, those that received a valid value (the general's private value) and those that did not receive any value during round one (undecided). At the beginning of round two, processes with a valid value choose it as their public value and attempt to send *valid-valued* messages to all other processes. Also at the beginning of the second round, the undecided processes send *"I don't know"* messages to all other processes.

From the second round onwards, the flow of the protocol becomes regular. There is only one event that differentiates the second and further rounds from the first one: If at the end of a round, there is a process that either received a null value from some process or *"I don't knows"* from all other processes, it then takes a null to be its public value, and sends out null messages to all other processes at the beginning of the following round. The protocol executes in this fashion until either all processes have decided on their public value or K rounds have occurred. Additionally, at the conclusion of the K^{th} round, all processes that are still undecided assume null as their public value. At the conclusion of the message exchanges, a decision about consensus can be taken irreversibly and can result in one of the following three results:

1. Valid Consensus: All the processes alive at the time of evaluating consensus had decided on a valid public value.

2. Null Consensus: All the processes alive at the time of evaluating consensus had decided on a null public value.

3. No Consensus: If neither valid- nor null-valued consensus could be arrived at, *no consensus* occurs. This condition occurs if there is a mix of valid-valued and null-valued processes, or if there are no live processes left.

3 Model

In this section, we develop a model to compute the probability of valid and null consensus under realistic fault conditions.

3.1 Computation and Fault Assumptions

The synchrony assumption is often not realistic, and we have thus allowed it to be violated stochastically. When the synchrony assumption is violated, messages are delayed and therefore not delivered within the time allowed for the progress of the round (called the *round length*[2] and denoted by R). This divergence from synchrony is called a *performance failure* [18]. We assume that whenever a performance failure occurs, the delayed message is lost.

We also allow for the possibility of messages getting lost due to the inherent unreliability of the underlying network (called an *omission failure* [18]). To quantify this behavior, we specify the probability that a message that is sent is lost, which we denote by q.

Performance and omission failures can cause a process to receive valid as well as null messages from the other processes, a condition not considered by protocol designers. For example, consider a system of five processes, with the general (p_1) broadcasting to each of his four lieutenants (p_2, p_3, p_4, p_5) at the beginning of round one. Assume that at the end of round one, p_2 received a valid value, while none of the others did. During round two, p_2 will broadcast a valid-valued message to all other processes, while each of the undecided processes will broadcast "*I don't know*" messages to all processes other than itself. Further, assume that at the end of round two, p_3 is the only process to have received a valid value successfully from p_2, p_4 received only "*I don't know*" messages sent by p_5 (and hence decided on a null public value), and p_5 itself is still undecided. Then, during the third round, p_3 will broadcast valid-valued messages while p_4 will broadcast null-valued messages, thus making it possible for p_5 to receive both kinds of messages at the conclusion of round three. To resolve this ambiguity, we assume that if a process receives valid as well as null-valued messages from other processes, it chooses the valid one as its public value.

Conversely, by introducing time explicitly into our model of the protocol's behavior, certain events that are possible in the theoretical description become extremely improbable. In particular, the theoretical protocol description assumes that processes can crash during the infinitesimal interval of time during which messages are sent out by a process in a single round. Though possible, the probability of this event is very small. Consequently, we also assume that the general cannot crash before initiating the broadcast to all his lieutenants at the beginning of the protocol's execution.

We assume that processes can fail anytime during the execution of the protocol except the infinitesimal interval of time when they are in the process of sending their messages, and we refer to the act of a process failing as a *crash failure*. We assume that crashed processes will no longer send messages, and that they will therefore not exhibit Byzantine behavior. Further, no repair mechanism is assumed, since typical repair times would be many times the order of the time allowed to reach a consensus decision.

Finally, we assume that a sending process needs to be alive only at the time of sending for the message to be sent. Consider a process p_i attempting to send a message to process p_j during round number r; the only requirement made of p_i in order to have a successful message delivery is that it be alive at the beginning of round r. This is a reasonable assumption, since the message transmission mechanism is assumed not to involve any kind of acknowledgment.

Consider now an attempt by a process p_i to send a message to another process, p_j, during round r. The message transmission would be initiated at the beginning of round r only if the process p_i were alive at that time. Assume further that the process p_j is also alive at the beginning of round r. Then, any one or more of the following will prevent p_j from receiving the message sent by p_i:

1. The receiving process, p_j, crashes before the end of the round r. It is correct to require the receiving process to be alive until the end of the round r, and not just until the instant when it receives the message from p_i, since the process p_j that receives the message during round r may not transmit it before the beginning of the round r+1.

2. The message transmission from p_i to p_j is lost during the transmission.

[2] This time is variously called the "round length," "round time," "clock-tick time," etc.

State	*Interpretation*
I	Crashed
II	Decided on valid value at least 2 rounds prior to the current round
III	Decided on null value at least 2 rounds prior to the current round
IV	Decided on valid value in the immediately previous round
V	Decided on null value in the immediately previous round
VI	Undecided

Table 1: Interpretation of the Six Process States.

3. The network was not able to deliver the message to p_j within R, the duration of a round.

3.2 Process State

As we argue below, six different process states are necessary to uniquely capture the current status of each process executing the considered consensus algorithm (see Table 1). Each process participating in the consensus algorithm must be in one of these states at any time. Further, as a consequence of the inherent round-based flow of the protocol, a process can make a transition from one state to another only at the boundary between consecutive rounds. We now explain the notion and significance of each of these six states.

If the process has crashed, it will not play any role in the successive rounds of message exchange. We denote a crashed process by state I. States II through VI describe the status of those processes that have not crashed. If a process has not crashed, it can be in one of three possible situations: it has a valid public value, it has a null public value, or it is still undecided. The remainder of this section describes the states that are necessary to capture these three situations.

Among those processes that have already decided on their public value, we need to differentiate between those that decided upon this value in the immediately preceding round, and those that did so two or more rounds prior to the current round. The reason for this distinction follows directly from the protocol specification. Processes that decide on a public value transmit it to all other processes and then stop, playing no further role in the message exchange process. In terms of the process-state model, this observation implies that only those processes that decided on a public value in the immediately preceding round play an active role in the current round. However, it is also important to maintain the state of processes that have decided on a value and stopped. Even though these processes do not exchange further messages, information about their status is required at the instant when the decision on whether consensus was reached is made, since the consensus decision depends on the number and state of processes that have not crashed. This requirement calls for four different states with the following interpretations:

Current State	*Next State*		
	I	*II*	*III*
I	1.0	0	0
II	p_{crash}	1 - p_{crash}	0
III	p_{crash}	0	1 - p_{crash}
IV	p_{crash}	1 - p_{crash}	0
V	p_{crash}	0	1 - p_{crash}

Table 2: Probabilities of Process State Transitions.

1. State II: The process decided on a valid value two or more rounds prior to the current round.

2. State III: The process decided on a null value two or more rounds prior to the current round.

3. State IV: The process decided on a valid value in the immediately previous round.

4. State V: The process decided on a null value in the immediately previous round.

Besides the five states already described, state VI signifies a process that has still not decided on a public value.

3.3 State Transitions and Probabilities

In this section, we use the six process states introduced in the previous section and enumerate all valid process-state transitions. Tables 2 and 3 enumerate the probabilities (Table 2) and the logical conditions that must hold (Table 3) for all valid process-state transitions during each round. Transitions shown with a zero probability of occurrence and those not shown are not allowable according to the protocol description and our assumed fault model.

Table 2 gives the probabilities of transitions originating from states I through V. The first row in this table indicates the probabilities for possible next states originating from the current state I (crashed). In the absence of any repair mechanism, as assumed by us, a process in state I can only stay in that state. A process in state II (valid-valued public value, stopped) can either crash (move to state I) with probability p_{crash}, or it can continue to remain in state II with probability 1-p_{crash}. Here, p_{crash} refers to the probability that a specified process that was not crashed at the beginning of the current round, has crashed before the end of that round. Similarly, a process in state III (null-valued public value, stopped) can either remain in its current state with probability 1-p_{crash}, or can move to state I with probability p_{crash}. Since a process in state IV received a valid value in the immediately preceding round, at the conclusion of the current round it will have held a valid value for two rounds, if it does not crash. Consequently, the options for a process in this state are either to crash (transition to state I) with probability p_{crash}, or to make the transition to state II (valid-valued public value, stopped) with probability 1-p_{crash}. Equivalently, a process in state V (null-valued public value, active) is allowed to make the transition

Next State	*Conditions for state transition from state VI to possible next states*
I	Process crashes during the current round.
II, III	Transition not possible.
IV	Process does not crash during the current round *And* At least one of the processes that sent out valid values (processes in state IV) during this round sent its message successfully to this process.
V	Not final round Process does not crash during the current round *And* None of the transmissions from any of the state IV processes to this process are successful *And* (At least one of the processes that were in state V at the beginning of the current round was able to deliver a null message successfully to this process *Or* The only messages received by this process during the current round were the "*I don't know*" messages sent to it by the other processes in state VI at the beginning of this round.) Final round Process does not crash during the current round *And* None of the transmissions from any of the state IV processes to this process are successful.
VI	Not final round Process does not crash during the current round *And* There was no successful transmission to this process from any of the processes that were in the states IV or V at the beginning of the current round *And* Process does not receive an "*I don't know*" message from any of the other processes that were in state VI at the beginning of the current round. Final round Transition not possible.

Table 3: Conditions for State Transition from State VI.

only to either state I, with probability p_{crash}, or state III, with probability 1-p_{crash}.

In Table 3, we consider the logical conditions that must hold for process-state transitions from state VI. The first row of Table 3 shows that one possibility is for this process to crash (state I), with probability p_{crash}. If it does not crash, it might receive a message (valid-valued, null-valued or "*I don't know*") from one of the other processes. If at least one of the messages received by this process was valid-valued (from state IV processes), then the process moves to state IV (row 2 in the table). Recall the assumption that in the event that a process receives a valid as well as a null value, it chooses the valid value as its public value. Consequently, the transition from state VI to state IV does not require the absence of any successful null message deliveries (from state V processes). If the state VI process neither crashed nor received any valid-valued messages, it can either move to state V (row 3), or remain in its current state, VI (row 4). These two transitions must each be broken into two sub-cases to account for the fact that all processes that are undecided at the end of the final round are forced to assume a null as their public value. In all rounds other than the final round, a state VI process that has not received any valid-valued messages would move to state V if it received at least one null-valued or "*I don't know*" message. During the final round, the absence of valid-valued messages is sufficient to make the process move to state V. Finally, this process would remain in state VI only if it did not receive any kind of message (valid, null, or "*I don't know*") from the other processes. However, a process can not remain in state VI at the end of the final round.

Symbol	*Represents*
M	Mean transmission time per message
λ	1/Mean lifetime of a process
R	Deterministic time allowed for the execution of a round
q	Probability of a message getting lost (omission failure)
ξ	Order of the Erlangian distribution for message delay times
Num_i	Number of processes in process state i at the beginning of the current round

Table 4: Process State Transition Probabilities

We now derive the transition probabilities of process-state transitions from state VI. The logical expressions presented in Table 3 hold true independent of the distribution assumed for process failure and message transmission times. However, the probabilities assigned to these events depend on the process failure and message delay distributions. Computation of these probabilities is possible for many distributions. For the purpose of illustration, we choose exponentially distributed process lifetimes, and exponential as well as Erlangian message delay times. Erlangian message delay distribution was chosen to study the effect of changes in variance on the behavior of the protocol.

We will assume the notation given in Table 4. There are three events whose probabilities are direct functions of the stochastic model assumed: process crash, omission failure and performance failure. The probability of a process crash with exponentially distributed lifetimes is $p_{crash} = 1 - e^{-\lambda * R}$, due to the memoryless property of the exponential distribution. The probability of an omission failure is q. The probability of avoiding a performance failure is $P[T \leq R]$, where T is the message transmission time. For the exponential case, the expression for $P[T \leq R]$ is given by

$$P[T \leq R]_{Exponential} = 1 - e^{-\frac{R}{M}}.$$

Next, consider an Erlangian distribution of order ξ such that the resultant mean transmission times are the same as in the case of the exponential distribution. This requires the common mean (S) of each of the ξ exponential stages that comprise the Erlang, to be $S = (M/\xi)$. The following expression gives the value of $P[T \leq R]$ in the Erlangian case:

$$P[T \leq R]_{Erlangian} = 1 - e^{-\frac{R}{S}}(\sum_{i=0}^{\xi-1} \frac{(\frac{R}{S})^i}{i!}).$$

We observe from Table 3 that all the logical conditions pertaining to message transmissions have one of the following two forms: a specified process receives at least one of the Num_i transmissions directed at it, or

a specified process does not receive any of the Num_i transmissions directed at it. Assuming message transmission times and omission failures to be independent, Equation 1 gives the probability that none of the Num_i transmissions from type i processes directed at a particular process are successful; the complement of this probability would be an expression for at least one of those transmissions being successful.

$$\phi_i = (1 - (1 - q) * P[T \leq R])^{Num_i} \quad (1)$$

Now consider the transition $VI \rightarrow IV$, and let Num_{IV} be the number of processes currently in state IV. The probability of the second clause of the logical condition enumerated in the table would then be $(1 - \phi_{IV})$. The probability of this transition, assuming process crashes, performance and omission failures to be independent, is

$$P[VI \rightarrow IV] = (1 - p_{crash}) * (1 - \phi_{IV}).$$

Next, consider the transition $VI \rightarrow V$. In the case of a non-final round, the probability of the process receiving no messages from state IV processes is ϕ_{IV}, and that of it receiving at least one message from state V processes is $1 - \phi_V$. The probability that this process will receive one or more "*I don't know*" messages (from state VI processes) is $1 - \phi_{VI}$, and the probability that it will receive only one or more "*I don't know*" messages (and no other messages) is $\phi_{IV}\phi_V(1 - \phi_{VI})$. Thus, the probability of the process transition from state VI to state V in any non-final round, which we denote $P[VI \rightarrow V]_{NotFinal}$, is given by

$$P[VI \rightarrow V]_{NotFinal} =$$

$$(1 - p_{crash}) * (\phi_{IV}) * [(1 - \phi_V) + \phi_V(1 - \phi_{VI})].$$

In the case of the final round, the expression for this probability simplifies to

$$P[VI \rightarrow V]_{Final} = (1 - p_{crash}) * \phi_{IV}.$$

Finally, the probability that a process in state VI continues to remain in that state, during any non-final round, is the probability that none of the transmissions to this process from the other processes in states IV, V or VI are successful. This probability is given by

$$P[VI \rightarrow VI] = (1 - p_{crash}) * (\phi_{IV}\phi_V\phi_{VI}).$$

3.4 State and Transition Probabilities

Given the state transition probabilities for an individual process, it is easy to construct the state of the entire group of processes. Specifically, the system state can be represented as a vector of dimensionality equal to the number of processes n. Formally, the system state can be written as:

$$S = (s_1, s_2, ..., s_n),$$

$$\text{where } s_i \in \{I, II, III, IV, V, VI\},\ i = 1 \text{ to } n,$$

where s_1 is the state of the general, and $s_2, s_3, ..., s_n$ are the states of the lieutenants. In order to understand how the system state evolves over time, we need to characterize the state transition probabilities between system states. To do this, we note that the events that drive state changes in the system (process, omission and performance failures) are independent of one another, given a particular system state. The transition probability from one state to another is thus the product of the probabilities of the transitions of individual processes in the group, i.e.,

$$P[R \rightarrow V] = \prod_{i=1}^{n} P[r_i \rightarrow v_i]. \quad (2)$$

Calculate_Consensus_Prob(*Current System State*: **C**, *Round Number*: **R**)
While there are unexplored next system states from **C**
Do Begin
Generate a new next system state **N**
Let State_Probability(**N**) = State_Probability(**C**)*
P[**C** → **N**]
Increment **R**
If **R**=K
Test for Consensus using the condition in Figure 3
If **Valid-valued Consensus** was declared
Increase Valid_Consensus_Prob by State_Probability(N)
Else If **Null-valued Consensus** was declared
Increase Null_Consensus_Prob by State_Probability(N)
Else **Calculate_Consensus_Prob**(**N, R**)
End Do

Figure 2: Algorithm for Calculating the Probability of Consensus.

3.5 Consensus Probability Calculation

We calculate the probability of consensus by building a tree of states reachable from the initial state, and summing up the probabilities along all those paths that lead to consensus. A node in this tree is a system state vector and each level of the tree corresponds to a different round of the protocol in execution. Consequently, any particular node in this tree at level i represents a possible state of the system of processes at the end of a round i in the protocol. Further, we store the probability of reaching each node at the node; this represents the probability of the protocol leading to the physical situation modeled by that node. We denote this probability as State_Probability(**C**) for node **C**.

At the root of this tree is the vector representing the state of the system before the protocol is set into execution. In this state, all the lieutenant processes are in state VI and the general is in state IV (since the general plays the same role during round one as another process in state IV would in any other round). Further, the probability value stored at the root is 1.0, since the protocol begins in that state with certainty. In Figure 2, we present a recursive algorithm to calculate the probability of valid-valued and null-valued consensus by generating the sequence of states reachable from an initial state, **C**. The invocation of this algorithm as **Calculate_Consensus_Prob**(**C** = Root, **R** = 0)

If
At least one process in either state II or state IV *And*
No processes in either state III or V *And*
The specified resiliency is not exceeded
Declare **Valid-Valued Consensus**
Else If
At least one process in either state III or state V *And*
No processes in either state II or IV *And*
The specified resiliency is not exceeded
Declare **Null-Valued Consensus**
Else Declare **No consensus could be arrived at**.

Figure 3: Procedure to Decide on Consensus.

results in the calculation of the appropriate consensus probabilities.

Given any node in the tree, there is a simple way of enumerating the next generation of nodes. Since a breadth-first approach would be extremely memory-intensive, we use the depth-first approach for traversing the probability tree. For the depth-first approach, the worst case memory requirements at any time are on the order of the height of the tree, which in turn is K, the maximum number of rounds. As pointed out earlier, a node in the tree is a vector representing the state of the n processes. Each of these processes, in a particular state, can make a transition only to a fixed set of new process states, with probabilities calculable as explained in Sub-section 3.3. By allowing each process to assume, in turn, all possible next process states, the algorithm enumerates all the possible next system states. In the algorithm of Figure 2, $P[\mathbf{C} \rightarrow \mathbf{N}]$ refers to the probability of a transition from system state **C** to system state **N**, and is calculated by the application of Equation 2.

In this algorithm, consensus is tested when K rounds have elapsed. The logical condition that dictates the result of testing this condition will result in one of three possible outcomes: valid-valued consensus, null-valued consensus, or no consensus. If explicit resiliency constraints are not desired, the clauses for resiliency in Figure 3 should be omitted.

4 Results

Though the model is solvable for any values of the system parameters, for our analysis, we consider system parameter values such that the probability of valid-valued consensus is at least 0.999999.

The distributed system executing the BG algorithm can be completely specified in terms of n, λ, R, M, K and q if the message delay distribution is exponential. Additionally, in case of an Erlangian message delay distribution, the order of the Erlang message delay distribution is specified. Recall that the algorithm of Figure 1, as defined, assumes that no more than m=n-3 processes may crash during the execution of the protocol. In the first study, we discuss why the decision on whether to enforce these resiliency requirements depends on the application in which the algorithm is used, and the system parameter values considered. We then present a study to investigate the number of rounds for which the protocol should be allowed to progress.

Resiliency	*Valid Consensus*	*Null Consensus*
Ignored	0.99999903	3.2e-22
Enforced	0.99999887	2.4e-22

n	K	R(ms)	M(ms)	λ(per sec)	q
4	2	200	17	2.5e-6	1e-7

Table 5: Variation in the Probability of Consensus With and Without Enforcing Resiliency Constraints. (Message Delay Times are Assumed to be Exponential.)

Next, we study the effect of variation in λ, M and q on the probability of valid consensus. We then analyze issues specific to Erlangian message delay times. We conclude this section by presenting a study to informally characterize the complexity of our model.

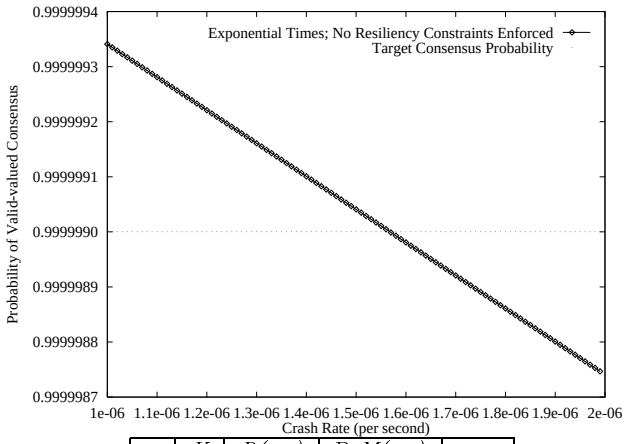
Enforcing/Not Enforcing Resiliency The requirement that the number of crashed processes not exceed the resiliency of the protocol makes sense only when the assumptions about the reliable, synchronous network hold. In that case, the requirement ensures that a consensus can always be reached at the end of m+1 rounds, where m=n-3 is the resiliency of the protocol. However, the usual definition of consensus does not require any minimum number of processes not to have crashed. In fact, [11] and others have analyzed how consensus can be achieved with non-negligible probability even when the number of crashes exceeds the resiliency of the protocol. In Table 4, we present the results for a case where ignoring resiliency constraints leads to the target consensus probability (0.999999) being achieved, but where enforcing them would have led to unacceptable behavior. The probability of null-consensus is extremely small due to the extreme unlikeliness of the general crashing (one of the preconditions for null consensus) at the extremely low process crash rates considered. Due to its insignificant contribution to the probability of consensus, null-valued consensus will not be dealt with in the remaining studies.

Varying the Number of Rounds Executed Once the assumptions about a reliable, synchronous network are violated, any claim about a solution to the BG problem needs to be restated in probabilistic terms. Therefore, it no longer makes sense to require the execution of the protocol through K=m+1 rounds [19]. We investigate the effect of variation in K on the probability of valid-valued consensus and find that, corresponding to any given set of system parameters, namely n, M, R, λ and q, there is an optimal value of K. Increasing K up to this value leads to an increase in the probability of consensus; any further increase in K leads to a decrease in the probability of consensus. This is because any increase in the consensus probability expected with an increase in K is more than offset by the requirement that the valid-valued processes (especially

K	*Probability of Valid Consensus*
1	0.73323824
2	0.99999960
3	0.99999940
4	0.99999920
5	0.99999900

n	$R(ms)$	$M(ms)$	$\lambda(per\ sec)$	q
6	200	40	1e-6	1e-5

Table 6: Variation in the Probability of Consensus with Variation in K. (Message Delay Distribution is Exponential and Resiliency Constraints are Not Enforced.)

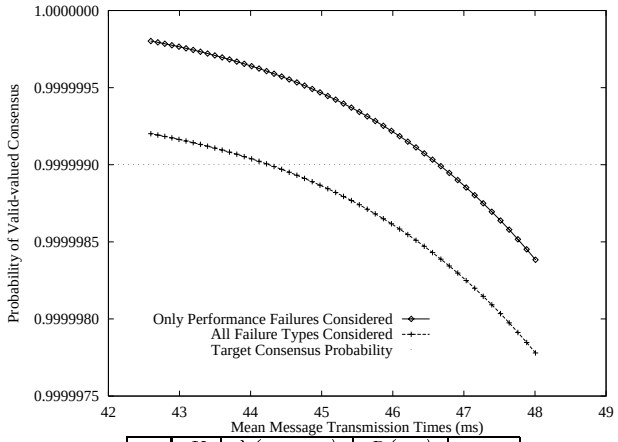


n	K	$R(ms)$	B $M(ms)$	q
5	3	200	40	1e-5

Figure 4: Variation in the Probability of Valid Consensus with the Process Crash Rate. (Resiliency Constraints are Not Enforced.)

the general) avoid crashing for an additional period of time. The value of K where the effect of crashes starts to play the predominant role in deciding the probability of consensus is a function of the values of the particular system parameters. Table 6 presents the results of one such study. For the system parameter values considered in this study, the probability of valid-valued consensus is maximum for K=2. Recall that for n=6, the BG algorithm of Section 2 would specify execution for K=4 rounds. The optimal value of K (in our study) is significantly lower than 4 because the small mean message transmission times (compared to R) considered here present an extremely favorable messaging environment. This study thus shows the importance of setting K based on the stochastic specification of the system.

Varying Crash Rate In the next study, whose results are presented in Figure 4, we allow the process crash rate, λ, to be varied over a range that allows for consensus to be achieved with the target value of



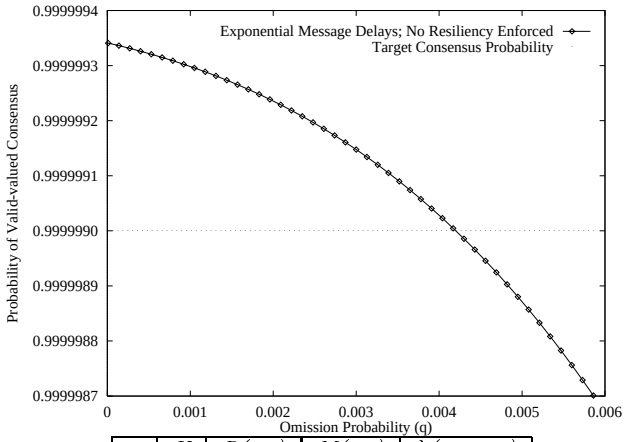
n	K	$\lambda(per\ sec)$	$R(ms)$	q
5	3	1e-6	200	1e-6

Figure 5: Variation in the Probability of Valid Consensus with the Mean of the Exponential Message Delay Distribution. (Resiliency Constraints are Not Enforced.)

0.999999. Nominal values are assigned to M and q, and consensus is evaluated in a system of 5 processes by allowing progress for K=3 rounds. The plot reveals that the maximum value that the process crash rate, λ, can attain and still allow for consensus to be achieved with the target value, is 1.6e-6. The plot also shows that the variation of consensus probability is largely linear with λ in this region. This is so because the (exponential) expression for the probability of process crashes (p_{crash}) takes a linear form for the very small values of λ we consider.

Varying Mean Message Delay Times We now consider the variation in M while holding the round length, R, at a fixed value (Figure 5). Besides analyzing the effect of variation in M with all types of failures allowed (crash, performance and omission), we have also studied this variation in the absence of crash and omission failures to analyze the influence of each component on the probability of consensus. It is interesting to observe that as the message delay times become progressively less favorable (M increases), the two curves start to converge, thus indicating that performance failures are starting to play the predominant role in this region of operation. The upper bounds on M to obtain a 0.999999 probability of valid-valued consensus in the two cases, for the parameter values considered, are, respectively, 44.24 ms (all failures) and 46.51 ms (only performance failures).

Varying Omission Failure Probability In Figure 6, we study the effect of varying q on the probability of valid-valued consensus. For the system parameters indicated in the accompanying table, the maximum permissible value of q that still allows the target consensus value to be achieved, is 0.00417. The variation



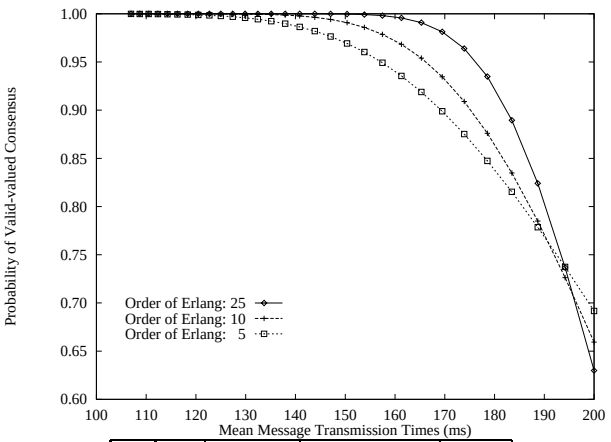
n	K	$R(ms)$	$M(ms)$	$\lambda(per\ sec)$
5	3	200	40	1e-6

Figure 6: Variation in the Probability of Valid Consensus with the Probability of Omission Failures. (Resiliency Constraints are Not Enforced.)

in valid-consensus probability with q has a polynomial form, since q occurs in terms of the form $(...q...)^{Num_i}$ (Equation 1).

Erlangian Message Delay Times In the previous three studies, we have restricted ourselves to exponential message delay times. In Figure 7, we consider Erlangian message delay times and study the effect of order of the Erlang distribution on the maximum value of M that still allows for consensus to be achieved with our target value. As is to be expected, an Erlang distribution of a higher order allows for more relaxed requirements on the messaging mechanism (tolerates higher values of M) in the region of interest. It is interesting to note the nature of the variation depicted by the three plots when the value of M tends to R, the round-time.

Complexity of Algorithm Finally, we present some results on the complexity of our model and the BG algorithm, per se, for a particular set of specifications. We have explicitly calculated the number of states generated in the corresponding probability tree for various system parameter values and believe that this number is a good indication of the complexity of the problem for a particular set of specifications. The number of states is a function of only n, K, and the decision to enforce or not enforce explicit resiliency requirements. In Table 7, we illustrate the effect of varying K, and whether or not the resiliency requirement is enforced, on the number of states. In Table 8, we capture the variation in the number of states with the number of processes, n. Observe here that n=3 is the lowest number of processes for which the problem is non-trivially defined. For each value of n, we assume K=n-2, as specified by the theoretical description of Section 2. Further, we assume the absence of any explicit resiliency requirements.



n	K	$R(ms)$	λ $(per\ sec)$	q
5	3	200	1e-6	1e-5

$Order\ of\ Erlang$	$Max.\ allowable\ M(ms)$
5	86.2
10	106.95
25	129.87

Figure 7: Effect of the Order of the Erlang on the Maximum Acceptable Mean Message Delay Time. (Resiliency Constraints are Not Enforced.)

K	$Without\ Resiliency$	$With\ Resiliency$
1	162	120
2	5771	3033
3	65262	19350
4	379213	64581
5	1499262	164706
6	4390058	282968

Table 7: Variation in the Number of States with Increase in K for n=5. (Resiliency Requirements are Enforced in One Case and Relaxed in the Other.)

n	K	$Number\ of\ States$
3	1	18
4	2	536
5	3	50388
6	4	6794248
7	5	646123316

Table 8: Variation in the Number of States with n. (Resiliency Requirements are not Enforced.)

From Tables 7 and 8, we observe that the number of states grows exponentially with both n and K. Further, the decision to enforce or not enforce resiliency has a much more significant effect on the number of states for high values of K.

5 Conclusion

We have presented a novel approach to probabilistically verify round-based protocols. The approach enumerates possible states that can be reached during the execution of a protocol, and computes the probabilities of desired and undesired behaviors. To illustrate the approach, we studied a simple variant of the Byzantine General's problem, designed to tolerate crash failures, executing in an environment that also leads to performance and omission failures. The results obtained shed new light on the relative importance of the occurrence rates of the various types of faults on correct operation of the protocol, and show that consensus can often be reached with high probability in a small number of rounds, even when assumptions made in the logical proof of the protocol are violated. We also show that for certain fault environments, the probability of reaching valid-valued consensus actually drops after a certain number of rounds are executed, and that this number is less than the number of rounds prescribed by the traditional proof. While the approach was developed in terms of a particular variant of the BG protocol, it should be applicable to any round-based protocol whose set of process states is finite.

More generally, the results enable us to quantify precisely the common belief that such protocols can continue to operate when assumptions made in their proof fail. Since many distributed systems will operate in hostile, non-deterministic environments, probabilistic verification can thus provide a more accurate method of assessing the goodness of a particular algorithm and design.

6 Acknowledgment

The authors wish to express their sincere thanks to David Powell of LAAS-CNRS for his advice during various stages of this work.

References

[1] J. Turek and D. Shasha, "The many faces of consensus in distributed systems," *Computer*, vol. 25, pp. 8–17, June 1992.

[2] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the Association for Computing Machinery*, vol. 27, pp. 228–234, April 1980.

[3] F. Cristian, "Reaching agreement on processor-group membership in synchronous distributed systems," *Distributed Computing*, vol. 4, pp. 175–187, April 1991.

[4] N. A. Lynch, *Distributed Algorithms*. San Francisco, California: Morgan Kaufman Publishers, Inc., 1996.

[5] A. Galleni and D. Powell, "Consensus and membership in synchronous and asynchronous distributed systems," Tech. Rep. N-96104, LAAS, 1996.

[6] J. Rushby and F. von Henke, "Formal verification of algorithms for critical systems," *IEEE Transactions on Software Engineering*, vol. 19, pp. 13–23, January 1993.

[7] B. Hailpern and S. Owicki, "Modular verification of computer communication protocols," *IEEE Transactions on Communications*, vol. 31, pp. 56–68, January 1983.

[8] L. Gong, P. Lincoln, and J. Rushby, "Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults," in *Fifth Conference on Dependable Computing for Critical Applications (DCCA-5)*, (Urbana-Champaign, IL, USA), pp. 79–90, September 1995.

[9] N. Maxemchuck and K. Sabnani, "Probabilistic verification of communication protocols," *Distributed Computing*, vol. 3, pp. 118–129, July 1989.

[10] G. Florin, C. Fraize, and S. Natkin, "A new approach of formal proof: Probabilistic validation," in *Second Conference on Dependable Computing for Critical Applications (DCCA-2)*, (Tucson, AR, USA), pp. 154–161, February 1991.

[11] O. Babaoglu, "Stopping times of distributed consensus protocols: A probabilistic analysis," *Information Processing Letters*, vol. 25, pp. 163–169, May 1987.

[12] D. Powell, "Failure mode assumptions and assumption coverage," in *22nd Symposium on Fault-Tolerant Computing (FTCS-22)*, (Boston, MA, USA), pp. 386–395, July 1992.

[13] A. Galleni and D. Powell, "Towards a unified comparison of synchronous and asynchronous agreement protocols," Tech. Rep. 95409, LAAS, 1995.

[14] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 382–401, July 1982.

[15] L. Lamport and M. Fischer, "Byzantine generals and transaction commit protocols," Tech. Rep. Opus 62, SRI International, 1982.

[16] R. Schlichting and F. Schneider, "Fail-stop processors: An approach to designing fault tolerant computing systems," *ACM Transactions on Computer Systems*, vol. 1, pp. 222–238, August 1983.

[17] D. Powell, D. Seaton, G. Bonn, P. Verissimo, and F. Waeselynk, "The 'DELTA-4' approach to dependability in open distributed computing systems," in *Eighteenth Symposium on Fault-Tolerant Computing (FTCS-18)*, (Tokyo, Japan), pp. 246–251, June 1988.

[18] F. Cristian, H. Aghili, R. Strong, and D. Dolev, "Atomic broadcast: from Simple Message Diffusion to Byzantine Agreement," in *Fifteenth Symposium on Fault-Tolerant Computing (FTCS-15)*, (Ann Arbor, MI, USA), pp. 200–206, June 1985.

[19] P. D. Ezhilchelvan, "Early stopping algorithms for distributed agreement under fail-stop, omission, and timing faults," in *Sixth Symposium on Reliability in Distributed Software and Database Systems*, (Williamsburg, VA), pp. 201–212, March 1987.