Abortable Consensus and Its Application to Probabilistic Atomic Broadcast

Wei Chen

Microsoft Research Asia

Beijing Sigma Center
No. 49 Zhichun Road, Haidian District
Beijing, China, 100080
weic@microsoft.com

Abstract

This paper introduces the specification of abortable consensus in message passing systems with probabilistic message delivery behaviors to address the tradeoff between progress and agreement in asynchronous consensus. The paper presents an abortable consensus algorithm, proves its correctness, and shows how to configure the parameters of the algorithm to satisfy the explicit requirement on the tradeoff between progress and requirement. The probabilistic analysis to the algorithm is novel in that it covers all possible failures and asynchrony allowed by the system model rather than some simple case studies as conducted by most previous researches. The paper further shows how to apply abortable consensus to probabilistic atomic broadcast, and shows that abortable consensus provides stronger properties than probabilistic atomic broadcast.

Keywords: abortable consensus, probabilistic atomic broadcast, probabilistic analysis, fault tolerance, distributed algorithm

1 Introduction

1.1 Motivation — Tradeoff between Progress and Agreement

Asynchronous consensus is a well-recognized problem in the area of distributed computing. In an asynchronous message-passing distributed system subject to process and/or communication failures, asynchronous consensus specifies a problem in which each process proposes a value and all processes eventually reach a unanimous decision on one of the proposed values. Asynchronous consensus is typically defined by the following properties (e.g. [7]):

- *Uniform Validity*: If a process decides v then some process previously proposed v.
- *Agreement*: Correct processes (i.e., processes that do not crash) do not decide different values.
- *Termination*: Eventually all correct processes decide.

Asynchronous consensus is at the core of many important distributed agreement problems, such as atomic broadcast, leader election, atomic commit, group membership, and mutual exclusion. Despite its importance, the well-known *FLP impossibility result* [15] shows that consensus is not solvable in a purely asynchronous system with the presence of even one possible process crash. To circumvent this impossibility result, several techniques, such as partial synchrony model [11, 12], failure detector model [7, 6], and randomization [4, 9], were introduced.

Although asynchronous consensus has been extensively studied, there are still some issues left without satisfactory answers. This paper addresses an important issue concerning the tradeoff between progress and agreement in asynchronous consensus, as explained below.

The above consensus specification requires that all correct processes must always agree on the decision value. To satisfy this requirement of absolute agreement, the consensus algorithms may have to sacrifice the progress of the consensus in certain situations. In particular, when a process cannot communicate with majority of the processes, the process cannot make a decision. This is because in an asynchronous system, the process cannot reliably detect if other processes have failed or not, and to avoid disagreement, the process has to wait until the communication with a majority of processes resume. In the extreme case where majority of processes fail, all remaining processes have to be blocked indefinitely. No progress can be made and the system comes to a halt.[1] Therefore, unreliable communication channels and process failures may delay or even halt the progress of consensus, because consensus requires absolute agreement and does not tolerate any disagreement at all.

While many fault-tolerant systems, such as state-machine replication [23], do require absolute agreement, there are cases in which a small degree of disagreement is tolerable, and thus sacrificing progress over agreement may not always be the best solution. For example, consensus can be used to implement atomic broadcast [7], which guarantees the agreement on the total ordering of message deliveries. However, absolute agreement on the total order may not be required. For instance, Felber and Pedone [14] studied probabilistic atomic broadcast, in which it is enough to guarantee message ordering with a high probability. In this case, consensus should be able to take advantage of the tolerance of a small probability of disagreement to achieve faster progress.

[1]A majority of processes being correct is not required in asynchronous systems augmented with a perfect failure detector $\mathcal{P}$ or a strong failure detector $\mathcal{S}$ [7]. However, $\mathcal{P}$ and $\mathcal{S}$ requires that at least some correct process never be suspected, which is essentially a synchrony requirement. So these classes of failure detectors are not of the concern here.

In general, the current asynchronous consensus specification and its implementations emphasize unilaterally on agreement over progress. As an alternative, this paper proposes a more flexible version of asynchronous consensus that makes the tradeoff between progress and agreement explicit and adjustable, so that applications can control this tradeoff according to their specific needs. This new version of consensus includes the existing consensus as a special case, and thus rather than conflicting with the existing studies on consensus, it is a significant extension that provides more flexible control to the applications.

1.2 Main Results of the Paper

1.2.1 Introducing Abortable Consensus

To address the tradeoff between progress and agreement, this paper introduces the *abort* action to consensus. Intuitively, when a process cannot communicate with majority of processes, the process can choose to abort the consensus instead of deciding a value, in order to avoid waiting indefinitely for messages from the majority processes. This version of the consensus is called *abortable consensus*.

The abort action is a compromise to the absolute agreement of consensus. Instead of requiring every correct process deciding on the same value, abortable consensus allows that some of the correct processes abort the consensus while others decide. However, for those processes that do make decisions, they still have to agree on the same value. Therefore, by allowing the abort action as a compromise to the absolute agreement, abortable consensus provides chances for a process to terminate rather than to be blocked indefinitely, meanwhile it still maintains agreement for other processes that do decide.

To control the degree of potential disagreement allowed by the abort action, an *abort probability threshold* α is introduced in the specification to limit the probability that a process aborts in a consensus run. The threshold α can vary from 0 to any value less than 1. When α is zero, abortable consensus does not tolerate any abort actions, and it is reduced to the traditional asynchronous consensus. When α increases, abortable consensus is more tolerant to the abort action, and it will terminate the consensus faster. Therefore, by varying the abort probability threshold α, abortable consensus covers a spectrum of consensus specifications that provide different degrees of tradeoff between progress and agreement. One end point of the spectrum is the traditional consensus that does not compromise agreement at all for progress.

The benefit that abortable consensus provides to applications is its flexibility. If an application demands absolute agreement, then it can use abortable consensus with $\alpha = 0$, which is equivalent to traditional consensus. If an application can tolerate a certain degree of disagreement, it can specify this degree of disagreement as the abort probability threshold α, and use the abortable consensus to achieve faster progress. This paper provides a specific example of using abortable consensus to implement probabilistic atomic broadcast to achieve an appropriate and provable tradeoff between progress and agreement.

1.2.2 Abortable Consensus Algorithm and Its Analysis

The basic idea of using the abort action to terminate consensus faster is simple. Practical systems may already use some sort of abort actions, such as killing the thread that is blocked by consensus, to allow the systems to make progress. The challenging part, however, is to guarantee the abort probability threshold α. This requires a careful design and analysis of the consensus algorithm.

The paper presents an abortable consensus algorithm based on the rotating coordinator algorithm of [7]. The algorithm has several modifications in order to facilitate its probabilistic analysis. The analysis of the algorithm is based on the probabilistic network model (e.g. [10, 3, 8]), where message delays and message losses follow some probability distribution.

The analysis proves that the algorithm implements abortable uniform consensus (a stronger version of abortable consensus), and derives the abort probability threshold α from the parameters of the algorithm and the system. Moreover, it shows how to configure the parameters of the algorithm to satisfy the abort probability threshold α for any given α. This enables applications to specify the abort probability threshold α and configure the consensus algorithm accordingly based on their needs.

The probabilistic analysis of the algorithm is novel in that it covers all possible scenarios that are allowed by the system model, including process crashes, message delays, message losses, asynchronous progress of consensus rounds, and even the asynchronous behavior of the propose actions. Moreover, the performance metrics are both asynchronous round-based and time-based. This is in contrast to most of the existing performance analyses on asynchronous consensus (e.g. [22, 19, 17, 1]), which are only asynchronous round-based, and are only conducted for several simple cases (such as when no process crashes, or when all crashes have already occurred and the system becomes stable).

An analysis that considers all possible scenarios is important in practice because it provides insights to the performance of a fault-tolerant system under all circumstances instead of just a few good-case scenarios. Despite the complexity of the analysis, the final results produced in the paper are in simple closed forms that can be used directly. To the best of the author's knowledge, the analysis in this paper is the first all-inclusive analysis to any version of asynchronous consensus that considers all different kinds of failures and asynchrony allowed by the underlying system model.

1.2.3 Application to Probabilistic Atomic Broadcast

In asynchronous systems, consensus is shown to be equivalent to *atomic broadcast* [7]. Atomic broadcast specifies that all correct processes deliver the same set of messages in the same order [16]. Felber and Pedone [14] proposed *probabilistic atomic broadcast (PABCast)*, which does not require absolute agreement on either the set of delivered messages or the order of message deliveries. Instead, agreement on message deliveries and orders only needs to satisfy certain probability thresholds. Naturally, PABCast bears similarities to abortable consensus.

The current paper studies the relationship between abortable consensus and PABCast, shows how to implement PABCast using abortable consensus, and derives the probability thresholds of PABCast from the abort probability threshold α of abortable consensus. The paper further shows that abortable consensus provides more properties than those specified by PABCast. More precisely, abortable consensus is equivalent to an enhanced version of PABCast. This demonstrates the applicability and the strength of abortable consensus with respect to other distributed problems.

To summarize, the current paper has the following contributions: (a) it introduces the abortable consensus specification in the message passing model to address the tradeoff between progress and agreement in asynchronous consensus; (b) it presents an algorithm that implements abortable consensus, and shows how to configure the algorithm to satisfy an application's requirement on the abort probability threshold α; (c) it is the first to provide an all-inclusive and quantitative performance analysis to an asynchronous consensus algorithm that considers all possible failure and asynchrony scenarios allowed by the system model, yet the analytical results are kept in simple closed forms; and (d) it shows how to implement probabilistic atomic broadcast using abortable consensus, with provable probabilistic guarantees derived from the abort probability threshold α, and it shows that abortable consensus is equivalent to an enhanced version of PABCast.

1.3 Related Work

To the best of the author's knowledge, there is no previous work that introduces the abort action to the specification of consensus, or other asynchronous agreement problems, to address the tradeoff between progress and agreement. The abort action in atomic commit [20] is different, and the difference is further discussed in Section 3.

In terms of the performance analysis to the consensus algorithms, Keidar and Rajsbaum summarized the results in this area in their PODC'02 tutorial [18]. However, as the title of the tutorial suggested, most of the existing studies only analyzed the performance of consensus algorithms when there are no faults in the system. This limits the applicability of a performance analysis to a fault-tolerant system. Keidar and Rajsbaum also pointed out the weakness of the asynchronous round-based performance metric used in the existing performance analyses, and they posed an open question on what is a better performance metric. This paper enriches the current research on performance analyses to consensus by providing an analysis that considers all failure and asynchrony scenarios allowed by the system model. It also provides a time-based performance metric as a better alternative to the asynchronous round-based metric.

In terms of the probabilistic network models and probabilistic analysis, they have been used earlier in clock synchronization protocols [10, 3], and later for several other distributed problems(e.g. [2, 5, 13, 14, 8, 21, 24]). The studies in [5, 13, 14] mainly focus on the scalability issue of the reliable broadcast or multicast, and apply probabilistic analysis on the gossip-style protocols. In particular, the work in [14] studies PABCast, which can be implemented by abortable consensus as shown in this paper. The study on PABCast in [14] differs from the one in the current paper in several aspects: a) the work in [14] focuses on scalability, while this paper focuses on the tradeoff between progress and agreement; b) abortable consensus can be used to implement more properties than those of PABCast; and c) for the probability thresholds of PABCast, [14] provides either recursive formulas or simulation results, while this paper provides analytical results in closed forms.

The work in [8] studies failure detectors, one of the fundamental components in fault-tolerant distributed systems, and its quality-of-service guarantees under the probabilistic network model. The current paper can be viewed as a continuation of the research in [8]: it uses the similar model and it addresses another fundamental problem in fault-tolerant distributed systems, namely consensus, in the probabilistic network model.

The study in [24] also applies a probabilistic analysis to a consensus algorithm. Its probabilistic analysis differs from the analysis in the current paper in that a) it makes significant simplifying assumptions in the analysis, such as all processes start every consensus round at the same time, and all crashes occur at the boundary between two successive rounds; and b) the analytical result also seems to be more complicated than the results presented in this paper.

The rest of the paper is organized as follows. Section 2 defines the system model. Section 3 presents the specification of abortable consensus, and discusses some of its implications. Section 4 provides the algorithm that implements abortable uniform consensus. Section 5 proves the correctness of the algorithm by a probabilistic analysis, shows how to configure the parameters of the algorithm to satisfy the abort probability threshold α for any given α, and analyzes the performance of the algorithm under all circumstances. Section 6 studies the relationship between abortable consensus and PABCast. Finally, Section 7 concludes the paper and discusses a few future directions.

2 System Model

2.1 Probabilistic Network Model

The system considered in the paper consists of a set of n processes, $\Pi = \{1, 2, \dots, n\}$. Processes communicate with each other by message passing through a communication network, which is modeled as a complete graph with bidirectional links connecting every pair of processes. Message passing is asynchronous, that is, messages may be delayed without bound or may be lost. To facilitate quantitative analysis, the model assumes that message delay and message loss behaviors follow certain probabilistic distribution, but the actual probabilistic distribution may not be completely known. More specifically, each link ℓ connecting a process p to a process q is characterized by the following two parameters: 1) *message loss probability* $p_L(\ell)$, which is the probability that a message from p to q is dropped by link ℓ, and 2) *message delay* $D(\ell)$, which is a random variable with range $(0, \infty)$ representing the delay from the time a message is sent by p to the time it is received by q, under the condition that the message is not dropped by the link (borrowing the same terminologies from [8]). The network is not necessarily symmetric, that is, the probabilistic behavior of each link may be different. The message delay and loss behaviors of different messages are independent.

The model ignores the execution delays of local actions, including the send and receive actions, of the processes. This assumption could be justified as follows. If the local execution delays were considered, one would have to model the atomic steps of local executions as well as the delay behaviors between atomic steps. Such a model could easily get very complicated and very difficult to analyze. Moreover, in our distributed environment, a process cannot distinguish whether another process is slow in executing local steps or the message from that process is slow. Thus, to some extent local execution delays can be viewed as part of message delays. Therefore, explicitly modeling local execution delays introduces much complexity without providing more insight, so the paper chooses to ignore the local execution delays.

Time is treated as continuous with range from 0 to ∞. Each process has access to a local clock, which can be used by the process to time out on messages or other actions. Local clocks may be skewed from the real time and from each other, but for simplicity, they are assumed to be drift-free, i.e., local clocks run at the same speed as the real time. In practice, clock drift rate is usually very small (in the order of 10^{-6} [10]). Thus, clock drift is negligible for the duration of one run of consensus, which is at the level of seconds, or at most minutes.

2.2 Process Failure Model

Processes may fail by crashing, i.e., stopping all its actions including sending and receiving messages. For simplicity, the model does not include process recovery. A process is *correct* if it never crashes; a process is *faulty* if it is not correct. A *failure pattern* F describes when a faulty process crashes in each run of consensus. Formally, failure pattern F is a function from Π to $[0, \infty]$. For each process p, $F(p)$ denotes the time at which process p crashes in this failure pattern; if $F(p) = \infty$, it means p does not crash, i.e., p is correct.

Process crashes may occur at any time, even between two consecutive local actions on a process, although the model ignores the execution delays of local actions for analytical purposes. The model does not assume that process crashes can be predicted either deterministically or probabilistically. That is, the state of the system at any given time has no information whatsoever (deterministically or probabilistically) on the occurrence of future crashes. Moreover, the delay and loss behaviors of the messages that a process sends are independent of whether and when the process crashes.

There are several reasons why process crashes are not modeled probabilistically as message delays and losses. First, conceptually it distinguishes the network behaviors from process crash behaviors, and emphasizes that the quantitative results of the paper are based on the quantitative characteristics of the network, but not on any quantitative assumptions on process crashes. Second, technically if the model does not make assumptions on process crashes, the results obtained are stronger, in that they are also correct when process crashes do follow certain probabilistic distributions. Third, typically in practice, it is relatively easier to estimate and predict message delay and loss behaviors than to estimate or predict process crashes.

3 Specification of Abortable Consensus

In *abortable consensus*, each process proposes a value and eventually every correct process either decides a value or aborts. Abortable consensus is required to satisfy the following properties:

- *Uniform Validity*: If a process decides v then some process previously proposed v.
- *Agreement*: Correct processes do not decide different values.
- *Termination*: Eventually all correct processes either decide or abort.
- *α-Abortability*: There exists an $\alpha < 1$ such that for any failure pattern in which a majority of processes are correct, the probability that there exists some process that aborts in a run with the failure pattern is at most α.

The α in the α-Abortability property is called the *abort probability threshold*. The property allows that some process aborts the consensus while other processes decide, but it limits the probability of this situation by α when a majority of processes are correct. The Agreement property, on the other hand, requires that among the correct processes that decide, they still decide on the same value. A stronger version of abortable consensus, called *abortable uniform consensus*, further requires:

- *Uniform Agreement*: Processes (correct and faulty) do not decide different values.

Several important points about the above specification are further explained below.

Restricting abort probability only when a majority of processes are correct. In the probabilistic network model with a non-zero message loss probability or unbounded message delays, it is still necessary to have a majority of processes being correct to implement asynchronous consensus, as stated by the following proposition.

Proposition 1 *In the probabilistic network model, if the message loss probability is non-zero, or message delays have no upper bound, it is necessary that a majority of processes be correct to implement asynchronous consensus as defined in Section 1.1.*

A standard partition argument similar to the one given in [7] can be applied here to prove this proposition, because even though message passing follows certain probabilistic distribution, it is still allowed a positive probability that arbitrary messages are lost or delayed for an arbitrarily long time, which is the key to establishing the partition argument.

The above proposition implies that the abort action is expected when a majority of processes are faulty, because no algorithm can guarantee that all correct processes decide in this case. Therefore, the abort action should only be restricted when a majority of processes are correct, as stated in the α-Abortability property.

Restricting the abort probability per failure pattern. Property α-Abortability restricts the probability of the abort action to α for *each* failure pattern where a majority of processes are correct. One may suggest an alternative that restricts the abort probability for all failure patterns together, such as "the overall probability that a process aborts in some run where a majority of processes are correct is limited by α". This, however, requires that process crashes be probabilistic in order to sum the abort probabilities among all possible failure patterns. As explained in Section 2.2, analytical results that are not based on the probabilistic assumption on process crashes can also be applied to the situations where process crashes do follow probabilistic distributions. Therefore, the α-Abortability specified for each failure pattern is a stronger property than the alternative specified over all failure patterns.

Traditional asynchronous consensus and abortable consensus. When the abort probability threshold α is zero, no abort action is allowed when a majority of processes are correct. Of course, the specification still allows abort actions when a majority of processes are faulty. However, in asynchronous systems (even if message delays are probabilistic), processes cannot reliably determine if a process has crashed or not. So no algorithm can take advantage of a specification that allows the abort actions when a majority of processes are faulty. Therefore, for all practical purposes, when α is zero, abortable consensus is reduced to the traditional asynchronous consensus as defined in Section 1.1.

Distinction between abortable consensus and atomic commit. Even though both abortable consensus and atomic commit [20] have agreement properties and an abort action (non-blocking atomic commit also has a termination property parallel to the termination property of abortable consensus), they are different in several aspects. First, in atomic commit, abort could be one of the input values, while in abortable consensus, abort is not part of the input; it is only an output option intended for exceptional cases. Second, atomic commit requires that if all processes are correct and they all propose commit, they all eventually commit, and if any process proposes abort, all processes have to abort; while in abortable consensus, there is no requirement on when processes have to decide or when they have to abort, other than the probabilistic restriction on the abort action. Finally, atomic commit requires that the abort action always be agreed among all processes, while abortable consensus does not require agreement on the abort action. Therefore, abortable consensus and atomic commit are two different problems.

4 Algorithm for Abortable Uniform Consensus

The algorithm given in Figure 1 implements abortable uniform consensus. It is based on the rotating coordinator algorithm with failure detector $\diamond\mathcal{S}$ [7]. In particular, the algorithm is also a round-based rotating coordinator algorithm with the same mechanism for locking the decision value and a similar messaging structures as the one in [7]. However, several modifications are made either to allow progress even when a majority of processes are faulty, or to simplify the probabilistic analysis. These modifications include: (a) allowing the coordinator to skip the current round after waiting a long enough time without successfully gathering a majority of responses; (b) allowing a process to abort after executing too many rounds; (c) replacing the failure detector with a timeout mechanism; (d) replacing the separate reliable broadcast for decision propagation with a simple built-in propagation scheme; and (e) allowing every process to skip the current round and immediately join the higher round when receiving a message of a higher round.[2]

In the algorithm, the processes proceed in asynchronous rounds. Each round has a *coordinator* exchanging messages with other processes (called *participants*) in several phases in order to reach a final decision. The coordinator executes two phases — NEWROUND and NEWESTIMATE — in each round. In each of these two phases, the coordinator sends some messages to all participants, collects a certain type of response messages from $\lfloor n/2 \rfloor$ participants, and executes some local actions according to the responses. Note that the coordinator always collects the response from itself automatically, so together with the $\lfloor n/2 \rfloor$ responses from the participants, the coordinator always collects a majority of responses before it proceeds.

Each participant executes two or three phases — SKIP, ESTIMATE and ACK — in a round. In each of these phases, each participant sends a message to the coordinator, waits for a response from the coordinator, and then executes some local actions.

[2]Modifications (d) and (e) are similar to the schemes used in [1].

For every process p:

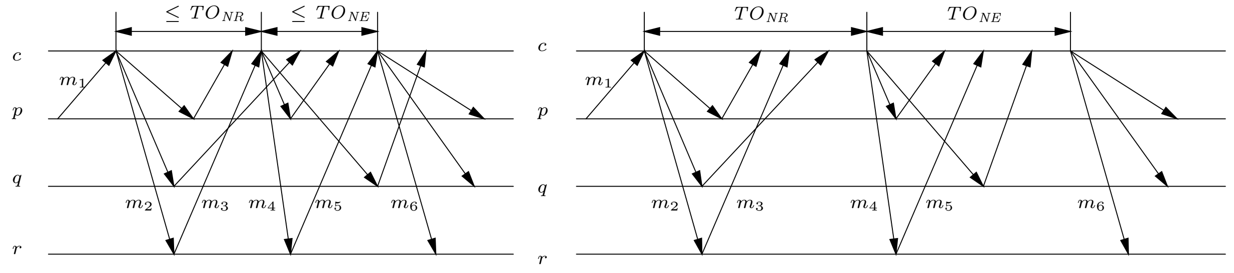
```
1  **upon** propose($v_p$):
2    $(r_p, estimate_p, ts_p) \leftarrow (1, v_p, 0)$                                   $\{$initialization$\}$
3    **if** $r_p > N$ **then output** abort; **return**                                 $\{$entry point of each round$\}$
4    $c_p \leftarrow (r_p \bmod n) + 1$                                                 $\{c_p$ is the coordinator of round $r_p\}$
5    **if** $p = c_p$ **then**                                                          $\{p$ is the coordinator of the current round $r_p\}$
6      **send** $(r_p,$ NEWROUND$)$ to all processes in $\Pi \setminus \{p\}$           $\{$begin Phase NEWROUND$\}$
7      **wait until** one of the following conditions is true, and execute the actions following the condition:
8        (1) [received $(r_p, estimate_q, ts_q,$ ESTIMATE$)$ from $\lfloor n/2 \rfloor$ participants] $\Longrightarrow$ **continue on** line 12
9        (2) [waiting time is longer than $TO_{NR}$ time units] $\Longrightarrow r_p \leftarrow r_p + 1$; **goto** line 3
10       (3) [received message $(r, \ldots)$ with $r > r_p$] $\Longrightarrow r_p \leftarrow r$; **goto** line 3
11       (4) [received message $(estimate,$ DECIDE$)$] $\Longrightarrow estimate_p \leftarrow estimate$; **output** decide$(estimate)$; **goto** line 23
12     $t \leftarrow$ largest $ts_q$ in $\{ts_q \mid p$ received $(r_p, estimate_q, ts_q,$ ESTIMATE$)$ **or** $q = p\}$
13     $estimate_p \leftarrow$ select one $estimate_q$ from
14       $\{estimate_q \mid p$ received $(r_p, estimate_q, t,$ ESTIMATE$)$ **or** $(q = p$ **and** $ts_p = t)$ $\}$
15     $ts_p \leftarrow r_p$                                                            $\{$end Phase NEWROUND$\}$
16     **send** $(r_p, estimate_p,$ NEWESTIMATE$)$ to all processes in $\Pi \setminus \{p\}$   $\{$begin Phase NEWESTIMATE$\}$
17     **wait until** one of the following conditions is true, and execute the actions following the condition:
18       (1) [received $(r_p,$ ACK$)$ from $\lfloor n/2 \rfloor$ participants] $\Longrightarrow$ **continue on** line 22
19       (2) [waiting time is longer than $TO_{NE}$ time units] $\Longrightarrow r_p \leftarrow r_p + 1$; **goto** line 3
20       (3) [received message $(r, \ldots)$ with $r > r_p$] $\Longrightarrow r_p \leftarrow r$; **goto** line 3
21       (4) [received message $(estimate,$ DECIDE$)$] $\Longrightarrow estimate_p \leftarrow estimate$; **output** decide$(estimate)$; **goto** line 23
22     **output** decide$(estimate_p)$                                                  $\{$end Phase NEWESTIMATE$\}$
23     **send** $(estimate_p,$ DECIDE$)$ to all processes in $\Pi \setminus \{p\}$; **return**   $\{$broadcast the decision to all processes$\}$
24   **else**                                                                           $\{p$ is a participant of the current round $r_p\}$
25     **if** not yet received $(r_p,$ NEWROUND$)$ from $c_p$ **then**
26       **send** $(r_p,$ SKIP$)$ to $c_p$                                              $\{$begin Phase SKIP$\}$
27       **wait until** one of the following conditions is true, and execute the actions following the condition:
28         (1) [received $(r_p,$ NEWROUND$)$ from $c_p$] $\Longrightarrow$ **continue on** line 33
29         (2) [received $(r_p, estimate_{c_p},$ NEWESTIMATE$)$ from $c_p$] $\Longrightarrow$ **goto** line 39
30         (3) [waiting time is longer than $TO_S$ time units] $\Longrightarrow r_p \leftarrow r_p + 1$; **goto** line 3
31         (4) [received message $(r, \ldots)$ with $r > r_p$] $\Longrightarrow r_p \leftarrow r$; **goto** line 3
32         (5) [received message $(estimate,$ DECIDE$)$] $\Longrightarrow$ **output** decide$(estimate)$; **return**   $\{$end Phase SKIP$\}$
33     **send** $(r_p, estimate_p, ts_p,$ ESTIMATE$)$ to $c_p$                          $\{$begin Phase ESTIMATE$\}$
34     **wait until** one of the following conditions is true, and execute the actions following the condition:
35       (1) [received $(r_p, estimate_{c_p},$ NEWESTIMATE$)$ from $c_p$] $\Longrightarrow$ **continue on** line 39
36       (2) [waiting time is longer than $TO_E$ time units] $\Longrightarrow r_p \leftarrow r_p + 1$; **goto** line 3
37       (3) [received message $(r, \ldots)$ with $r > r_p$] $\Longrightarrow r_p \leftarrow r$; **goto** line 3
38       (4) [received message $(estimate,$ DECIDE$)$] $\Longrightarrow$ **output** decide$(estimate)$; **return**
39     $(estimate_p, ts_p) \leftarrow (estimate_{c_p}, r_p)$                            $\{$end Phase ESTIMATE$\}$
40     **send** $(r_p,$ ACK$)$ to $c_p$                                                 $\{$begin Phase ACK$\}$
41     **wait until** one of the following conditions is true, and execute the actions following the condition:
42       (1) [received message $(estimate,$ DECIDE$)$] $\Longrightarrow$ **output** decide$(estimate)$; **return**
43       (2) [waiting time is longer than $TO_A$ time units] $\Longrightarrow r_p \leftarrow r_p + 1$; **goto** line 3
44       (3) [received message $(r, \ldots)$ with $r > r_p$] $\Longrightarrow r_p \leftarrow r$; **goto** line 3   $\{$end Phase ACK$\}$

45 **upon receive** $m$ **from** $q$:                                                    $\{$passive propagation of the decision value$\}$
46   **if** $m = (r, \ldots)$ **and** decide$(estimate)$ has occurred **then**
47     **if** $p = (r \bmod n) + 1$ **then**                                            $\{p$ is the coordinator of round $r\}$
48       **send** $(estimate,$ DECIDE$)$ to all processes in $\Pi \setminus \{p\}$     $\{$broadcast the decision to all processes$\}$
49     **else**
50       **send** $(estimate,$ DECIDE$)$ **to** $q$                                     $\{$send the decision to the coordinator $q$ of round $r\}$
```

Figure 1: Algorithm solving abortable uniform consensus

In the ideal cases when all the timings are right, all processes decide at the end of a round, which is the successful round as defined later in Section 5.1. Figure 2(a) depicts a typical successful round, as explained below. First, some process p enters the round and sends a SKIP message to the coordinator c of the round (line 26, phase SKIP). When the coordinator c receives the SKIP message, it jumps to this round immediately, and sends a NEWROUND message to every participant (line 6, phase NEWROUND). When a participant receives the NEWROUND message from c, it enters this round (if not yet so), and sends an ESTIMATE message with its current estimate of the decision value to the coordinator c (line 33, phase ESTIMATE). When the coordinator c has received ESTIMATE messages from at least $\lfloor n/2 \rfloor$ participants, it selects an estimate value with the latest timestamp, stores the new estimate value and its timestamp (lines 12–15, phase NEWROUND), and then it sends a NEWESTIMATE message with the newly selected value to every participant (line 16, phase NEWESTIMATE). When a participant receives the NEWESTIMATE message from c, it updates its local estimate and the timestamp (line 39, phase ESTIMATE), and then sends an ACK message back to c (line 40, phase ACK). When the coordinator c has received the ACK messages from at least $\lfloor n/2 \rfloor$ participants, it decides on its estimate value (line 22, phase NEWESTIMATE), and sends out the DECIDE message with the decision value to every participant before it returns from the consensus run (line 23). Finally, when every participant receives the DECIDE message from c, it decides on the value attached with the message and returns from the consensus run (line 42, phase ACK).

In less-than-ideal cases, message delays and losses and process failures may prevent processes from reaching a decision in a round. The algorithm uses several schemes to avoid blocking the progress of the processes in such cases. First, each of the five phases has a timeout so that no process is blocked in one phase forever. If a process times out in a phase, it directly goes to the next round (lines 9, 19, 30, 36, and 43). Second, if a process receives a higher round message, it jumps to that round immediately (lines 10, 20, 31, 37, and 44). This guarantees that processes respond to the latest round messages immediately, which is important to the probabilistic analysis. Third, if a process receives a DECIDE message, it decides immediately on the value contained in the message (lines 11, 21, 32, 38, and 42). If the process is the coordinator of the round, it further broadcasts the decision value to all participants before it returns (line 23). Fourth, after a process decides, it still helps other processes by sending out DECIDE messages whenever it receives a non-DECIDE message (lines 45–50). This guarantees that a temporarily disconnected process can still decide after it reconnects



(a) a succesful round of the abortable consensus algorithm (b) a benchmark run of algorithm A

m_1—SKIP message; m_2—NEWROUND message; m_3—ESTIMATE message; m_4—NEWESTIMATE message; m_5—ACK message; m_6—DECIDE message;

Figure 2: Comparison between a successful round of the abortable consensus algorithm and a benchmark run of algorithm A

with the rest of the processes that already decided. Finally, if a processes advances to a round higher than a threshold N, it gives up on reaching a decision and aborts (line 3). This guarantees the termination of the consensus run.

The algorithm has six parameters: the *abort round threshold* N, which is the number of the highest round that any process may enter before the process chooses to abort, and five timeout values, TO_{NR}, TO_{NE}, TO_S, TO_E, and TO_A, one for each of the five phases. A few obvious improvements to the algorithm are ignored to make the algorithm more understandable and easier for analysis.

5 Analysis of the Algorithm

5.1 Correctness of the Algorithm

To prove the correctness of the algorithm, one needs to show that the algorithm satisfies Uniform Validity, Termination, Uniform Agreement, and α-Abortability. This section focuses on the α-Abortability property and provides the main idea on how to analyze the abort probability threshold of the algorithm. The complete proof of correctness is included in Appendix A.

Definition 1 *Let a* successful round *in a run of the algorithm be the round such that there exists some process entering the round, and all processes entering the round either decide or crash in this round.*

The following lemma shows that some process aborting the consensus is directly related to the lack of a successful round.

Lemma 2 *Suppose there is at least one process that enters round* 1*. There is a process that aborts in a run of the algorithm if and only if there is no successful round in the run.*

For the purpose of analysis, consider the following fictitious algorithm A that resembles one round of the abortable consensus algorithm.

Definition 2 *Let c be a process called the coordinator, and other processes are called participants. Let p be one of the participants. Define algorithm A with c as the coordinator and p as the starting process as follows. The algorithm starts with process p sending a* SKIP *message to the coordinator c. When c receives the* SKIP *message from p, c sends a* NEWROUND *message to every participant. When a participant receives the* NEWROUND *message from c, it sends an* ESTIMATE *message to c. After c sends out the* NEWROUND *messages for TO_{NR} time units, c sends a* NEWESTIMATE *message to every participant. When a participant receives the* NEWESTIMATE *message from c, it sends an* ACK *message to c. After c sends out the* NEWESTIMATE *messages for TO_{NE} time units, c sends a* DECIDE *message to every participant.*

Definition 3 *Let a* benchmark run *of algorithm A be a run in which all processes are correct, and it satisfies the following conditions for every participant q: (a) the time elapsed from p sending the* SKIP *message to q receiving the* NEWROUND *message from c is at most TO_S; (b) the time elapsed from c sending the* NEW-ROUND *message to c receiving the* ESTIMATE *message from q is at most TO_{NR}; (c) the time elapsed from q sending the* ESTIMATE *message to q receiving the* NEWESTIMATE *message from c is at most TO_E; (d) the time elapsed from c sending the* NEWESTIMATE *message to c receiving the* ACK *message from q is at most TO_{NE}; (e) the time elapsed from q sending the* ACK *message to q receiving the* DECIDE *message from c is at most TO_A (see Figure 2(b)).*

A benchmark run resembles a clean successful round of the abortable consensus algorithm, with the following important difference: in a benchmark run, the coordinator always waits until the end of the timeout to send a new type of messages, while in a successful round of the abortable consensus algorithm, the coordinator sends a new type of messages to all participants as soon as it receives the expected messages from half of the participants (see Figure 2(a) for a comparison). The above point is used to establish a key result in the analysis of α-Abortability, which shows the relationship in probability between the successful rounds of the abortable consensus algorithm and the benchmark runs of the algorithm A, as explained below.

Definition 4 *Consider the runs of algorithm A in which all processes are correct, and p is the starting process and $c \neq p$ is the coordinator. Let $\beta(p, c)$ be the probability that among these runs a run of algorithm A is a benchmark run. Since the network may not be symmetric, the $\beta(p, c)$'s may be different with different pairs of p and c. Let* benchmark probability *β be the minimum among all $\beta(p, c)$'s.*

The major technical analysis is to establish the result that (roughly speaking) the probability of any round being successful when a majority of processes as well as the coordinator of the round are correct is at least the benchmark probability β (Lemma 15 in Appendix A). This is the key to the entire analysis, because it masks many complicated behaviors, such as process crashes, message losses and message delays that could occur in a successful round, and reduces all different possible cases of successful rounds into a simple and tractable type of benchmark runs. It is easy to see that given a reasonable set of timeout values, the benchmark probability β should be greater than zero. Then as the number of rounds increases, the probability of no round being successful, which is the same as the probability that some process aborts in the run, should decrease exponentially fast.

To complete the analysis, one also has to consider the asynchrony of the propose actions of the processes.[3] At one extreme, if all processes propose at the same time, the analysis is relatively simple, but it is an unrealistic synchrony assumption. At the opposite extreme, if there is no restriction whatsoever on when a process proposes, the abortable consensus becomes impossible, since the propose actions on a majority of processes can be delayed for an arbitrarily long time such that a minority of processes that proposed early always abort, even though all processes are correct and eventually propose. To make the proposal asynchrony tractable, the probabilistic restriction is applied to the delay in the propose actions.

Definition 5 *Suppose that a majority of processes are correct. Let* proposal delay V *be a random variable representing the elapsed time from the time when the first process proposes to the time by which a correct majority of processes (processes that form a majority and are correct) have proposed. The probability that* V *is within a given value* t*, denoted by* $\Pr(V \le t)$*, tends to* 1 *when* t *tends to* ∞*.*

The above probabilistic restriction on the propose actions is reasonable, since when a consensus algorithm is used as a component to solve other distributed problems, the propose actions on all processes are usually coordinated in some fashion. For example, the algorithm in [7] that implements atomic broadcast with a consensus algorithm uses a reliable broadcast algorithm to coordinate the propose actions of the consensus run, and thus the proposal delay V can be derived from the analysis to the reliable broadcast algorithm in this case.

[3]If, when a process receives a consensus message before it proposes, it can immediately propose its own value and start its consensus run, then the analysis does not need to consider the asynchrony of the propose actions, and the proposal delay in the analytical results can be omitted. This, however, only applies to the situations where the processes are already running and already know their proposals and the propose actions are not triggered by the application.

The following theorem summarizes the result of the analysis and shows that the algorithm given is correct.

Theorem 1 *The algorithm given in Figure 1 satisfies Uniform Validity, Uniform Agreement, and Termination properties. Furthermore, if the benchmark probability β is nonzero, the algorithm with the abort round threshold $N = j+kn$, $j, k \in \{1, 2, 3, \ldots\}$, satisfies α-Abortability with $\alpha = \gamma+(1-\gamma)(1-\beta)^{k\lceil(n+1)/2\rceil} < 1$, where $\gamma = \Pr(V > jT_m)$, V is the proposal delay, and $T_m = \min(TO_{NR}, TO_{NE}, TO_S, TO_E, TO_A)$. Therefore, the algorithm implements abortable uniform consensus with α tending to zero as N tends to infinity.*

The α-Abortability given in the above theorem can be interpreted as follows. The abort probability threshold is divided into two parts: a) the probability that some process aborts before a correct majority of processes have proposed, and b) the probability that some process aborts after a correct majority of processes have proposed. For a), γ is an upper bound on this probability, and j is the number of initial rounds needed to achieve the bound γ. For b), first, $(1-\gamma)$ is the probability that no process aborts when a correct majority of processes have proposed; second, $(1-\beta)^{k\lceil(n+1)/2\rceil}$ is an upper bound on the probability that some process aborts after a correct majority of processes have proposed, and kn is the number of remaining rounds needed to achieve this bound. Therefore, adding the two upper bounds, we reach an upper bound on the overall abort probability with $N = j + kn$ rounds.

5.2 Configure Algorithm Parameters Given α

This section shows how to calculate the parameters of the algorithm to satisfy any given α. This is important in practice because it allows applications to configure the algorithm according to their tolerance to the abort actions. The complete analysis is included in Appendix B.

Definition 6 *Given a timeout value TO, for any link ℓ, let $\lambda(\ell)$ be the probability that a message sent on link ℓ is delivered within TO time units after it is sent. Let* delivery probability *λ of a network for the timeout TO be the minimum of $\lambda(\ell)$'s for all links in the network.*

With the probabilistic network model defined in Section 2, one can write $\lambda = \min_\ell((1-p_L(\ell))\cdot\Pr(D(\ell) \leq TO))$. Note that, in practice, one does not need to know $p_L(\ell)$ and the entire distribution of $D(\ell)$ to get $\lambda(\ell)$. In fact, some simple experiments will give very good estimates on $\lambda(\ell)$ for a link ℓ. Given an appropriate TO, λ should always be greater than zero.

The following lemma gives a lower bound on the benchmark probability β using the delivery probability λ.

Lemma 3 *Suppose the delivery probability of a network for a given timeout TO is $\lambda > 0$. Set the timeout parameters of the abortable consensus algorithm as follows: $TO_{NR} = TO_{NE} = TO_S = 2TO$, and $TO_E = TO_A = 3TO$. Then the benchmark probability β has a lower bound λ^{5n-4}.*

Note that all timeout settings are not the same. This reflects the fact that the responses to NEWROUND, NEWESTIMATE and SKIP messages are immediate, while the responses to ESTIMATE and ACK messages are not — the coordinator needs to wait for $\lfloor n/2 \rfloor$ messages before sending out a response.

The following theorem shows how to configure the parameters of the algorithm to satisfy the α-Abortability for any given α, when the delivery probability λ of a network for a certain timeout TO is known, .

Theorem 2 *Suppose the delivery probability of a network for a given timeout TO is $\lambda > 0$. For any value $\alpha \in (0, 1)$, suppose $j \in \{1, 2, 3, \ldots\}$ is such that $\Pr(V > 2j\,TO) \leq \alpha/2$, where V is the proposal delay. Then the algorithm in Figure 1 satisfies α-Abortability for the given α if the parameters of the algorithm are set up as follows: $TO_{NR} = TO_{NE} = TO_S = 2TO$, $TO_E = TO_A = 3TO$, and $N = j + kn$ where*

$$k = \left\lceil \frac{\log(\alpha/2)}{\lceil (n+1)/2 \rceil \log(1 - \lambda^{5n-4})} \right\rceil .$$

To illustrate the application of Theorem 2, consider a simple example as follows. Suppose that in a network with 10 processes, the probability that a message is delivered within 10ms on any link is at least .99, i.e., $n = 10$, and the delivery probability of the network is $\lambda = .99$ for timeout $TO = 10ms$. Suppose the requirement on the α-Abortability of the algorithm is that the probability that some process aborts when a majority of processes are correct is at most .0001, i.e. $\alpha = .0001$. Plugging these numbers into the formula for calculating k above, one obtains $k = 2$. That is, after initial j rounds so that a correct majority of processes propose, $2n = 20$ more rounds are needed for the algorithm to guarantee that $\alpha = .0001$. Since each process spends at most $\max(TO_{NR} + TO_{NE}, TO_S + TO_E + TO_A) = 8TO$ time units in one round, the algorithm will terminate at most $20 * 8 * 10 = 1600ms$ after the initial j rounds.

In terms of the number j of the initial rounds needed to ensure that a correct majority of processes have proposed, it depends on the context in which the algorithm is used. If all processes propose at the same time, then $j = 0$. If all processes propose within T time units, then it is enough to set $j = T/(2TO)$. If the propose actions of all processes are coordinated by some other distributed protocols, such as reliable broadcast, then the distribution of the proposal delay V is likely to be close to some exponential distribution, in which case a moderate value j should be good enough to satisfy $\Pr(V > 2j\,TO) \leq \alpha/2$.

An important result given by Theorem 2 is that the relationship between α and k is a log-relationship. That is, a much stronger requirement on α only requires a small increase in k to satisfy it. For example, in the above numerical example, if the requirement on α is strengthened from .0001 to .00001, the computed k only increases from 2 to 3.

Furthermore, when timeout TO increases, the delivery probability λ usually increases, and it leads to smaller values for j and k, i.e. less rounds are needed to achieve α-Abortability. On the other hand, as TO increases, each round takes more time to complete. Therefore there is a tradeoff between using a longer timeouts in each round and using more rounds to achieve α-Abortability. With the result in Theorem 2, it is possible to compute an optimal TO so that the total time each process spends on one consensus run is minimal.

5.3 The Special Case: $N = \infty$

The algorithm has a special case where $N = \infty$. In this case, the algorithm continues running until all processes decide, and no process ever aborts. Obviously, the abort probability threshold α is zero. The Uniform Validity and Uniform Agreement properties still hold. For the Termination property, Theorem 1 shows that as the number of round tends to infinity, the probability that some process has not decided tends to zero. Thus, when $N = \infty$, all correct processes eventually decide with probability one. So the algorithm with $N = \infty$ satisfies the following property:

- *Probability-One Termination*: For any failure pattern, with probability one eventually all correct processes decide.

Technically, the above property is not exactly the same as the Termination property of asynchronous consensus given in Section 1.1, but for all practical purposes, they do not have essential difference. Therefore, the algorithm with the special case $N = \infty$ implements asynchronous consensus (with the Termination property replaced by the Probability-One Termination property).

5.4 Performance of the Algorithm

The analysis of the algorithm leads to several performance metrics of the algorithm. Suppose the algorithm uses the settings as given by Theorem 2. First, since each process spends at most $8TO$ time units in one round, the longest time any process may take in one run of the algorithm is $8(j + kn)TO$, where j and k are determined by Theorem 2.

Second, and more interestingly, is to derive the expected number of rounds and the expected time the algorithm spends before reaching a decision or abort, when a majority of processes are correct. From Lemma 3, one knows that the probability of a successful round is at least λ^{5n-4}, given that a majority of processes as well as the coordinator of the round are correct. Let a *correct round* denote a round in which the coordinator is correct, and a *faulty round* denote a round in which the coordinator is faulty. Thus, when a majority of processes are correct, after the processes pass the initial j rounds such that a correct majority of processes have proposed, the expected number of correct rounds needed is at most $1/\lambda^{5n-4}$. Suppose there are $f < n/2$ faulty processes. For every n consecutive rounds, there are f faulty rounds and $n - f$ correct rounds. Let $f_0 = f\lceil 1/(\lambda^{5n-4}(n-f))\rceil$. So among the $1/\lambda^{5n-4}$ correct rounds, there are at most f_0 faulty rounds. Therefore, the expected number of rounds for any process to complete one run of the algorithm (after the initial j rounds) is at most $f_0 + 1/\lambda^{5n-4}$, and the corresponding expected time to complete one consensus run is at most $8TO(f_0 + 1/\lambda^{5n-4})$. Using the same numerical example given in Section 5.2 and let $f = 4$, the expected number of rounds is at most 5.59, and the expected running time is at most $447ms$.

Note that the above analysis only considers that eventually there is always a round that is successful, which means N is infinity and no process ever aborts. This is the reason why the expected values are only affected by TO, λ, f and n, but is not affected by α. With N being a finite value, no process will go beyond round N, so the expected number of rounds for termination should be smaller. This is exactly the tradeoff that abortable consensus is providing: with some probability of the abort actions, processes may terminate the algorithm faster. The following is an informal analysis on this tradeoff provided by the algorithm.

For simplicity, assume that there are a correct majority of processes and they have proposed. Given an N, suppose that the actual probability that a process may abort is α_0, which must be less than α. Let R_N be the expected number of rounds that the algorithm takes to terminate the consensus, by either reaching a successful round by round N or aborting at the end of round N. Let R_∞ be the special case of R_N where $N = \infty$. Let R_N^- be the expected number of rounds that the algorithm takes to terminate the consensus, given the condition that the algorithm reaches a successful round by round N. The probability that the above condition is true is $(1 - \alpha_0)$. Let R_N^+ be the expected number of *additional* rounds that the algorithm has to take to reach a successful round after round N, given the condition that it does not reach a successful round by round N and it continues without ever aborting the consensus. The probability that the above condition is true is α_0. Therefore, we have the following equations: $R_\infty = (1-\alpha_0)R_N^- + \alpha_0(N + R_N^+)$, and $R_N = (1-\alpha_0)R_N^- + \alpha_0 N$. Since R_N^+ is the expected number of the additional rounds needed after round N, and in each round processes run the same protocol and require the same condition to have a successful round, R_N^+ is essentially the same as R_∞.[4] Thus, we have $R_N = (1-\alpha_0)R_\infty$.

The above argument shows that by compromising a certain degree of agreement (allowing an abort probability α_0), abortable consensus does gain in progress, i.e., early termination of the consensus, by a factor of $(1-\alpha_0)$; and the higher the abort probability, the faster the termination of the consensus.

Finally, message complexity can also be obtained. In each round (except the rounds in which the coordinator already decides), at most $6(n-1)$ messages are exchanged in the network, so the total number and the expected number of messages exchanged in one run of the algorithm can be derived using the above results on the total number and the expected number of rounds.

It is important to point out that the above performance metrics apply to all consensus runs with all possible failure and asynchrony scenarios allowed by the system model, and they provide both round-based and time-based performance results. In contrast, most of the existing researches only provide performance analyses that are asynchronous round-based and are limited to several simple cases. Moreover, the above performance results are simple and can be easily applied in practice, despite the apparent complexity to consider all possible failure and asynchrony scenarios.

6 Application to Probabilistic Atomic Broadcast

6.1 Specification of Probabilistic Atomic Broadcast

In atomic broadcast, each processes broadcasts a number of messages (could be infinite) and messages have to be delivered on all correct processes in the same order. As shown in [7], atomic broadcast is equivalent to consensus in asynchronous systems. Recently, Felber and Pedone [14] propose probabilistic atomic broadcast (PABCast), in which the set and the order of the messages delivered by each process only need to agree with each other with certain probability thresholds. This specification allows them to use gossip-style protocols to achieve good scalability.

Our paper demonstrates that abortable consensus can be used to implement PABCast such that the proba-

[4]The argument omits some details such as the number of crashes may be different before or after round N, which may cause the R_N^+ and R_∞ not being the same. However, the argument is intended as a simple and intuitive discussion on the performance gain with the abort action, so such omissions should be tolerable.

bility thresholds of PABCast can be analytically derived from the abort probability threshold α of abortable consensus. It will further show that abortable consensus can provide stronger properties than PABCast.

The specification of PABCast is given by the following properties, which are essentially the same as the ones given in [14].

- *Integrity*: Every message is delivered at most once at each process, and only if it was previously broadcast.
- *Probabilistic Agreement*: There exists a $\gamma_a > 0$ such that for any failure pattern in which processes p and q are correct, if p delivers message m, then the probability that q delivers m is at least γ_a.
- *Probabilistic Validity*: There exists a $\gamma_v > 0$ such that for any failure pattern in which process p is correct, if p broadcasts message m, then the probability that p delivers m is at least γ_v.
- *Probabilistic Order*: There exists a $\gamma_o > 0$ such that for any failure pattern in which processes p and q are correct, if p and q both deliver m and m', then the probability that they do so in the same order is at least γ_o.

In order to implement PABCast using abortable consensus, another component, probabilistic reliable broadcast, is needed. This is introduced in the next section.

6.2 Probabilistic Reliable Broadcast

Probabilistic reliable broadcast (PRBCast) is another broadcast specification that requires the Integrity, Probabilistic Agreement, and Probabilistic Validity properties as defined in Section 6.1. So the difference between PRBCast and PABCast is that PRBCast does not require the Probabilistic Order property.

Figure 3 shows the algorithm that implements PRBCast, based on the algorithm in [16]. It is a basic flooding algorithm with repeated send actions (lines 8–11) to overcome possible message losses. To distinguish the broadcast and deliver primitives of PRBCast and PABCast, these primitives are prefixed with ’PR’ or ’PA’ whenever necessary, both in the algorithms and in the analyses.

The algorithm in Figure 3 is meant to show that PRBCast is implementable in the probabilistic network model. In practice, the algorithm could be replaced by more scalable algorithms such as the one given in [5].

The following theorem summarizes the correctness of the algorithm, and its proof is given in Appendix C.

For every process p:

```
upon PR-broadcast$(m)$:
  PR-send$(m)$ to all processes excluding $p$
  PR-deliver$(m)$
upon PR-receive$(m)$ from $q$:
  if $p$ has not previously executed PR-deliver$(m)$ then
    PR-send$(m)$ to all processes excluding $p$
    PR-deliver$(m)$
upon PR-send$(m)$ to $q$:
  repeat at most $k$ times
    send $m$ to $q$
  until received $(ACK, m)$ from $q$
upon receive $m$ from $q$:
  send $(ACK, m)$ to $q$
  if $p$ has not previously executed PR-receive$(m)$ then PR-receive$(m)$
```

Figure 3: Implementing PRBCast

Theorem 3 *The algorithm in Figure 3 implements probabilistic reliable broadcast in the probabilistic network model, with the probability thresholds $\gamma_v = 1$ and $\gamma_a = 1 - p_L^k$, where p_L is the maximum of message loss probability $p_L(\ell)$'s for all communication links ℓ in the system, and k is the maximum number of repeated send actions on a message as described in the algorithm.*

6.3 Implementing PABCast with Abortable Consensus

With PRBCast and abortable consensus, one can implement PABCast, as shown in Figure 4. The algorithm is based on the atomic broadcast algorithm using consensus in [7], with one important addition. The basic idea of the algorithm is to use PRBCast to deliver the messages to the processes, and then use abortable consensus to agree on the delivery order. The addition to the original algorithm is on how to treat the abort actions of abortable consensus: If a process aborts, it will atomically deliver the messages proposed by itself (line 15). This addition is an important factor that affects the analytical results on the probability thresholds of PABCast.

The following theorem summarizes the correctness of the algorithm and provides the probability thresholds it satisfies. The complete analysis is given in Appendix D.

Theorem 4 *Suppose the probabilistic reliable broadcast algorithm used in Figure 4 has γ_a^R and γ_v^R as the thresholds for the Probabilistic Agreement and Probabilistic Validity properties, respectively. Suppose the abortable consensus algorithm used in Figure 4 has α as the threshold for the α-Abortability property. The PABCast algorithm given in Figure 4 satisfies the Integrity property, the Probabilistic Agreement property with*

For every process p:

```
Initialization:
  $R\_delivered \leftarrow \emptyset$; $A\_delivered \leftarrow \emptyset$; $k \leftarrow 0$;

upon PA-broadcast($m$):
  PR-broadcast($m$)                                  {to PABCast a message, PRBCast it first}

upon PR-deliver($m$):
  $R\_delivered \leftarrow R\_delivered \cup \{m\}$

when $R\_delivered - A\_delivered \neq \emptyset$:
  $k \leftarrow k + 1$;
  $A\_undelivered \leftarrow R\_delivered - A\_delivered$
  propose($k, A\_undelivered$)                       {call abortable consensus to decide on the delivery order}
  wait until decide($k, msgSet^k$) or abort
  if decided on a value ($k, msgSet^k$) then
    $A\_deliver^k \leftarrow msgSet^k - A\_delivered$      {if decided on a set of messages, they are the ones to be PA-delivered next}
  else                                               {the $k$-th consensus is aborted}
    $A\_deliver^k \leftarrow A\_undelivered$               {if aborted, messages in my own proposal are the ones to be PA-delivered next}
  PA-deliver($m$) for all $m \in A\_deliver^k$ in some deterministic order
  $A\_delivered \leftarrow A\_delivered \cup A\_deliver^k$
```

Figure 4: Implementing PABCast using abortable consensus

threshold $\gamma_a^A = \gamma_a^R(1-\alpha)$, the Probabilistic Validity property with threshold $\gamma_v^A = \gamma_v^R\gamma_a^R(1-\alpha)$. If a majority of processes are correct, it also satisfies the Probabilistic Order property with threshold $\gamma_o^A = (1-\alpha)^2$. Therefore, the algorithm implements probabilistic atomic broadcast when a majority of processes are correct.

The probability thresholds of PABCast given in the above theorem have the following intuitive interpretations:

- For the probability threshold γ_a^A on the Probabilistic Agreement property, for two correct processes p and q to PA-deliver a message m, it is sufficient that both processes PR-deliver m (with probability at least γ_a^R), and then both processes decide in a consensus run with m as part of the decision (with probability at least $(1-\alpha)$), so γ_a^A is at least $\gamma_a^R(1-\alpha)$.[5]

- For the probability threshold γ_o^A on the Probabilistic Order property, to PA-deliver two messages m and m' in the same order, it is sufficient that the two messages are part of the decision values of two consensus runs with no abort actions at all, and thus γ_o^A is at least $(1-\alpha)^2$.

- For the probability threshold γ_v^A on the Probabilistic Validity property, the interpretation may not be as

[5]This interpretation is still a simplified version, because the use of the abort probability threshold α requires that a majority of processes be correct, but the result $\gamma_a^A = \gamma_a^R(1-\alpha)$ does not require that. For a complete argument, see the proofs of Lemmata 20 and 21.

straightforward. For a correct process p to PA-deliver a message m that it has previously PA-broadcast, it is sufficient that p first PR-delivers m (with probability at least γ_v^R), and then p PA-delivers m after a consensus run. The probability that p PA-delivers m given that p PR-delivers m is proven to be at least $\gamma_a^R(1-\alpha)$, which may not be immediately intuitive (see the proof of Lemma 22 in Appendix D). Therefore, γ_v^A is at least $\gamma_v^R\gamma_a^R(1-\alpha)$.

The theorem shows that abortable consensus can be used to implement PABCast. Furthermore, the probability thresholds of PABCast can be easily derived from the abort probability threshold α of abortable consensus.

6.4 Enhancement to PABCast

The above section shows that abortable consensus can be used to implement PABCast. However, the algorithm in Figure 4 does not fully utilize the potential of the abortable consensus specification. In particular, when a process p aborts in a consensus run, it merely PA-delivers all messages it proposes for this consensus run (line 15 of Figure 4), as if those messages are actual decision values of the consensus run. But process p is aware that it does not make a decision, and thus the messages it proposed are likely not the ones that other processes will decide on in the same consensus run. Thus, process p could add a flag to each of these messages indicating that they are potentially out of order. These *out-of-order flags* provide further information to applications about orderings of the messages.

With the out-of-order flags, it is easy to see that the enhanced algorithm guarantees that all messages PA-delivered prior to the first message with the out-of-order flag on each process must follow the same order. That is, the first out-of-order flag on each process marks the boundary before which the messages are guaranteed to be PA-delivered in the same order. Furthermore, the α-Abortability also guarantees that the probability that the out-of-order flag is attached to a message is at most α, when a majority of processes are correct. The above is summarized by the following two properties.

- *Pre-Flag Complete Order*: For two correct processes p and q, and for the two sequences of messages that p and q deliver before their first out-of-order flags respectively, one sequence is the prefix of the other sequence.

- *Flag Restriction*: There exists an $\alpha < 1$ such that for any failure pattern in which a majority of processes are correct, the probability that with this failure pattern there exists some process that delivers its first message with an out-of-order flag is at most α.

The Pre-Flag Complete Order property also has a uniform version, which does not require p and q to be correct processes. If the abortable consensus satisfies uniform agreement, then the corresponding implementation of PABCast satisfies the uniform version of the Pre-Flag Complete Order property. Moreover, the Pre-Flag Complete Order is a stronger property than an ordering property that only requires any two pre-flag messages delivered by p and q following the same order. For example, it excludes the possibility of process p delivering messages m_1, m_2 and m_3 while q delivers m_1 and m_3, before their first out-of-order flags respectively.

It will be shown in this next section that the above two properties are important in distinguishing the strength between abortable consensus and PABCast.

Note that ordering is only guaranteed for messages before the first out-of-order flag on a process. After the first out-of-order flag, even if some messages are not attached with the out-of-order flags, they may not be delivered in the same order as in other processes. For example, suppose there are three messages m_1, m_2 and m_3 to be delivered by three runs of abortable consensus. In the first run, every process except process p decides m_1, while process p aborts and delivers on its own proposal m_2. So after the first consensus run, p PA-delivers m_2 with an out-of-order flag, while others PA-deliver m_1 without the flag. In the second consensus run, every process including p decides m_3, and so they all PA-deliver m_3 without the flag. In the third consensus run, p proposes m_1 since it has not delivered m_1 yet, and all processes decide m_1. Thus p PA-delivers m_1 without the flag, but others skip m_1 since they already PA-delivered m_1 before. The result is that p PA-delivers m_3 before m_1, while other processes PA-deliver m_1 before m_3, and none of the deliveries is attached with a flag. Therefore, after the first out-of-order flag, further ordering is not guaranteed by the absence of the flag, but the flags may still be useful in providing hints on which messages are likely to be out of order.

6.5 Relationship between Abortable Consensus and PABCast

The previous sections already shows that abortable consensus can be used to implement PABCast, which implies that abortable consensus is at least as strong as PABCast. Is the reverse true? That is, can abortable consensus be implemented using PABCast, and hence they are equivalent of each other?

Before we proceed to study this problem, one important issue needs to be clarified first. With the probabilistic network model, both abortable consensus and PABCast can be implemented. When studying the relative strength between two implementable specifications, one has to be careful in choosing the transformation algorithm to show that one can be implemented by the other. If there is no restriction on the transformation algorithm, of course any implementable specification can be "transformed" from any other specification — just let the transformation algorithm itself implement the specification. Such transformations are useless in studying the relative strength between two specifications.

Therefore, to study the relative strength between two implementable specifications, we need to use restricted transformations that are not strong enough to implement the specifications by themselves. Studying restricted transformations in asynchronous distributed systems is an interesting research topic by itself, but the full treatment is out of the scope of this paper. For the purpose of understanding the relationship between abortable consensus and PABCast, this paper uses a simple type of restricted transformations — *silent transformations*, which means that the transformation itself does not involve sending or receiving any messages. Since such transformations cannot be used to implement any meaningful distributed specifications by themselves, they are good candidates to be used to study the relationship between two implementable specifications.

In fact, the algorithm in Figure 4 is a silent transformation from abortable consensus to PABCast, built on top of the probabilistic network model and PRBCast. That is, besides the PRBCast part, the algorithm only calls abortable consensus and there is no more message exchanges needed by the algorithm. Therefore, as long as PRBCast itself is not as strong as PABCast, which is not hard to believe, the algorithm indeed shows that abortable consensus provides enough properties to implement PABCast.

For the remaining discussion, we will also use silent transformations to study whether PABCast is good enough to implement abortable consensus.

In the case of consensus and atomic broadcast, implementing consensus using atomic broadcast is very simple [11]. To propose a value, a process atomically broadcasts it. To decide a value, a process picks the value of the first message that it atomically delivered. Note that this is a silent transformation, since it only calls atomic broadcast and no messages are exchanged by the transformation algorithm itself.

This silent transformation is however not good enough for PABCast to implement abortable consensus. There are two problems that prevent the above simple transformation to work. First, in PABCast, the first messages delivered by processes are not necessarily the same due to the Probabilistic Agreement and Probabilistic Order properties. So if processes decide on their first delivered messages, it may violate the Agreement property of abortable consensus. But if a process does not decide a value, there is no further information that a process can deduce in order to abort the consensus with a guaranteed abort probability threshold α. Therefore, it is reasonable to suspect that the PABCast specification as given in Section 6.1 is not strong enough to solve abortable consensus.

This problem can be circumvented with the two enhancement properties provided in Section 6.4. With the enhancement properties, if a process delivers the first message with an out-of-order flag, then it aborts the consensus; otherwise, it decides on the first message. The Pre-Flag Complete Order guarantees that the first messages without an out-of-order flag are the same, and thus processes have to decide on the same value — Agreement property is satisfied. The Flag Restriction property guarantees that when a majority of processes are correct, the probability that a flag appears in the first message is at most α, and thus the probability that any process aborts is at most α — α-Abortability is satisfied.

Even with the two enhancement properties, it is still not enough to implement abortable consensus. The specification of PABCast allows a positive probability that a correct process never delivers any message. If a correct process does not even deliver one message, it certainly cannot decide on any value. The process cannot abort either, because the PABCast specification has no indication on how long it will wait before aborting the consensus run to guarantee a small probability of abort. But if a correct process neither decides nor aborts, it violates the Termination property of abortable consensus.

To circumvent this problem, yet another property is needed for PABCast:

- *Guaranteed Progress*: if a correct process broadcasts a message, then eventually it delivers some message.

With the Guaranteed Progress property, a process has to deliver some message after broadcasting a message, and thus depending on whether it delivers the first message with an out-of-order flag, the process either decides or aborts — Termination property is satisfied. It is easy to verify that the PABCast algorithm using abortable consensus given in Figure 4, together with the PRBCast implementation given in Figure 3, satisfies the Guaranteed Progress property.

Therefore, with the three enhancement properties (Pre-Flag Complete Order, Flag Restriction, and Guaranteed Progress), there is a silent transformation that transforms the enhanced PABCast to abortable consensus. Moreover, all these enhancement properties can be satisfied by an implementation of PABCast using abortable consensus. Hence, abortable consensus is equivalent to PABCast enhanced by all three properties stated above.

It is interesting to note that, with the introduction of the above three enhancement properties, abortable consensus can be implemented by PABCast without using the Probabilistic Agreement, Probabilistic Validity or Probabilistic Order properties in the original specification of the PABCast. These three properties have been superseded by the new properties introduced.

The following theorem summarizes the relationship between abortable consensus and PABCast.

Theorem 5 *Abortable consensus is equivalent to PABCast enhanced with the Pre-Flag Complete Order, Flag Restriction, and Guaranteed Progress properties. Moreover, the above three properties together with the Integrity property of PABCast are enough to implement abortable consensus.*

7 Concluding Remarks

This paper studies abortable consensus to address the tradeoff between progress and agreement in asynchronous consensus. It also shows the application of abortable consensus to probabilistic atomic broadcast and shows their relationships. There are several possible future research directions based on this research work. One is to further improve the algorithm and performance analysis to provide better performance guarantees. More importantly, the tradeoff between progress and agreement is also an important issue for many other distributed problems, so the techniques introduced in this paper can potentially be applied to these problems as well. Finally, this research, together with the one in [8] and possible others following the same direction, may provide the basic components and analytical tools that could be used to build fault-tolerant distributed systems with well-studied performance and quality-of-service guarantees.

References

[1] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13(2):99–125, Apr. 2000.

[2] C. Almeida and P. Veríssimo. Timing failure detection and real-time group communication in quasi-synchronous systems. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, June 1996.

[3] K. Arvind. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):475–487, May 1994.

[4] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, Aug. 1983.

[5] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

[6] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996. An extended abstract appeared in *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing*, August, 1992, 147–158.

[7] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, Mar. 1996. A preliminary version appeared in *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, Aug., 1991, 325–340.

[8] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(5):561–580, May 2002.

[9] B. Chor and C. Dwork. Randomization in byzantine agreement. *Advances in Computer Research*, 5:443–497, 1989.

[10] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989.

[11] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, Jan. 1987.

[12] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, Apr. 1988.

[13] P. T. Engster and R. Guerraoui. Probabilistic multicast. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 313–324, June 2002.

[14] P. Felber and F. Pedone. Probabilistic atomic broadcast. In *Proceedings of the 21st Symposium on Reliable Distributed Systems*, pages 170–179, Oct. 2002.

[15] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, Apr. 1985.

[16] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. J. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.

[17] M. Hurfin and M. Raynal. A simple and fast asynchronous consensus protocol based on a weak failure detector. *Distributed Computing*, 12(4):209–223, 1999.

[18] I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults – a tutorial, July 2002. In Tutorial of 21st ACM Symposium on Principle of Distributed Computing (http://theory.lcs.mit.edu/ idish/ftp/podc02-tutorial.ppt).

[19] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[20] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[21] M. Raynal and F. Tronel. Group membership failure detection: a simple protocol and its probabilistic analysis. *Distributed Systems Engineering Journal*, 6(3):95–102, 1999.

[22] A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, 1997.

[23] F. B. Schneider. Replication management using the state-machine approach. In S. J. Mullender, editor, *Distributed Systems*, chapter 7, pages 169–198. Addison-Wesley, 1993.

[24] F. Tronel. A probabilistic analysis of the consensus problem (fast abstract). In *Proceedings of the 2000 International Conference on Dependable Systems and Networks*, pages B16–17, June 2000. Full version as technical report 1226, IRISA, by E. Fronmentin and F. Tronel, 1999.

Appendix

A Proof of Theorem 1

For completeness, the definitions, lemmata, and the theorems in the main text are restated here in the Appendices.

This appendix proves the correctness of the abortable consensus algorithm given in Figure 1. All line numbers in this appendix refer to the lines in Figure 1.

A process p *enters round* r if it finishes executing line 3 with $r_p = r$. Process p *decides in round* r if p enters round r and then outputs $\mathsf{decide}(est)$ for some est without entering any other round $r' \geq r$. Process p *aborting in round* r and *crashing in round* r are similarly defined.

Lemma 4 (Uniform Validity) *If a process decides v then some process previously proposed v.*

Proof. Trivial. □

Lemma 5 (Termination) *Eventually all correct processes either decide or abort.*

Proof. Due to the timeout mechanism in each phase of the algorithm, no correct process stays in any phase of the algorithm forever, and thus no correct process stays in any round of the algorithm forever. Therefore, a correct process either decides or eventually advances to a round higher than the round threshold N, in which case it aborts the consensus run. □

Lemma 6 *Suppose that the coordinator c of round r outputs $\mathsf{decide}(est)$ in line 22. In every round $r' \geq r$, if the coordinator c' updates $estimate_{c'}$ to some value est' in lines 13–14, then $est' = est$.*

Proof. The proof is done by induction on the round number r'. For the base case ($r' = r$), if c outputs $\mathsf{decide}(est)$ in line 22 of round r, then c must have updated $estimate_c$ with value est in lines 13–14 of the same round. Thus the base case is correct.

Now assume that the lemma holds for all $r', r \leq r' < k$. Let c' be the coordinator of round k. The following shows that the lemma holds for $r' = k$.

Suppose that in round k, c' updates $estimate_{c'}$ to some value est' in lines 13–14. Then c' has received messages of the form $(k, *, *, \text{ESTIMATE})$ from $\lfloor n/2 \rfloor$ participants (according to line 8). Let $P_1 = \{p \mid c'$ has received $(k, *, *, \text{ESTIMATE})$ from $p\} \cup \{c'\}$. Since c executes line 22 in round r, c has received (r, ACK)

from $\lfloor n/2 \rfloor$ participants (according to line 18). Let $P_2 = \{p \mid c$ has received (r, ACK) from $p\} \cup \{c\}$. Since $|P_1| = |P_2| = \lfloor n/2 \rfloor + 1$, it follows that $P_1 \cap P_2 \neq \emptyset$. Let $p \in P_1 \cap P_2$.

From $p \in P_2$, one has that either $p = c$ or c has received (r, ACK) from p. In either case, p has updated its estimate to the round-r coordinator's estimate est (lines 13–14 if $p = c$ or line 39 if $p \neq c$). Therefore, in round r, the value of p's timestamp variable ts_p is updated to r. From $p \in P_1$, one has that either $p = c'$ or c' has received $(k, *, *, \text{ESTIMATE})$ from p in round k. In either case, p enters round $k > r$. Since the value of ts_p is non-decreasing, when p enters round $k > r$, its ts_p value is at least r. Thus, when c' selects value t in line 12 in round k, t is at least as large as the value of ts_p at the time, i.e. $t \geq r$. It is easy to see that every ts_q participated in the selection of t in line 12 of round k has $ts_q < k$. So, $r \leq t < k$.

Let q be the process whose estimate value est' is selected by the coordinator c' in lines 13–14 of round k. Thus, q updates its estimate to est' in round t, either in lines 13–14 or in line 39. In either case, the coordinator c'' of round t updated $estimate_{c''}$ to est' in lines 13–14. Since $r \leq t < k$, by the induction hypothesis, one has $est' = est$. $\Box$

Lemma 7 (Uniform Agreement) *Processes (correct and faulty) do not decide different values.*

Proof. Suppose that processes p and p' decide on values est and est', respectively. Note that the only place in the algorithm where decision is not caused by receiving a DECIDE message is line 22. Thus, by a simple induction on the number of DECIDE messages that cause the decisions, there must be coordinators c and c' decide on values est and est' in line 22 of round r and r', respectively. Without loss of generality, assume $r' \geq r$. Since coordinator c' outputs $\mathsf{decide}(est')$ in line 22 of round r', it must have updated $estimate_{c'}$ to est' in lines 13–14 of round r'. By Lemma 6, $est' = est$. $\Box$

The rest of the section is for the probabilistic analysis on the α-Abortability property of the algorithm. The analysis presented below chooses to omit some mathematical details, such as the formal definitions on the runs of the algorithm, probability spaces, probability measures, and so on. It is enough to point out that the probability space underlying the probabilistic analysis would be the set of all runs of the algorithm given a certain failure pattern. Further details are not essential to the understanding of the algorithm and its analysis.

Definition 1 *Let a* successful round *in a run of the algorithm be the round such that there exists some process entering the round, and all processes entering the round either decide or crash in this round.*

Proposition 8 *If a process p enters round $r > 1$, then for all round r' with $1 \leq r' < r$, there must be some* *process p' that enters round r' and later enters round $r' + 1$ due to a timeout in one of the phases (lines 9, 19, 30, 36 and 43).*

Proof. According to the algorithm, process p enters a round $r > 1$ either by receiving a message of round r when it is still in a lower round, or by timing out in one of the phases of round $r - 1$ and advancing to round r (lines 9, 19, 30, 36 and 43). Thus, by a simple induction there is always a process that enters round $r - 1$ and later enters round r due to a timeout in one of the phases (lines 9, 19, 30, 36 and 43). The proposition can be obtained by repeating the above argument on round $r - 1, r - 2$, and so on. □

Proposition 9 *No process enters round $r > N$ in any run of the algorithm.*

Proof. Suppose, for a contradiction, that a process p enters round $r > N$. By Proposition 8, there is a process p' that enters N and later enters $N + 1$. However, right before entering $N + 1$, p' executes line 3 and since $r_{p'} > N$, p' aborts, and thus p' never enters round $N + 1$ — a contradiction. □

Proposition 10 *If round r is a successful round, then no process enters round $r' > r$. That is, the successful round is always the final round of the run of the algorithm.*

Proof. Suppose, for a contradiction, that a process p enters a round $r' > r$. Then by Proposition 8 there must be a process p' that enters round r and later enters round $r + 1$. However, since round r is a successful round, after entering round r, p' either decides or crashes in round r, and thus p' never enters round $r + 1$ — a contradiction. □

Proposition 11 *There is at most one successful round in every run of the algorithm.*

Proof. Immediate from Proposition 10. □

Proposition 12 *If there is a successful round in a run of the algorithm, then no process aborts in this run.*

Proof. Suppose, for a contradiction, that a process p aborts in the run. Since p can only abort at the end of round N, p enters round N. Let r be the successful round. By Proposition 9, $r \leq N$. If $r = N$, then by definition p either decides or crashes in round N, contradicting with the assumption that p aborts in round N. If $r < N$, then by Proposition 10, no process enters round N, again contradicting with the assumption that p enters round N. Thus, no process aborts in the run. □

Proposition 13 *For any $r \in \{1, 2, \ldots, N - 1\}$, if a process enters round r and round r is not a successful round, then there is a process that enters round $r + 1$.*

Proof. Since some process enters round r and round r is not successful, by definition there is a process p that neither decides nor crashes in this round. Since $r < N$, p cannot abort in this round either. Because p has timeout in each phase of the algorithm, it cannot stay in round r forever, so it must enter a higher round $r' > r$. By Proposition 8, there is a process p' that enters round $r + 1$. □

Proposition 14 *Suppose there is at least one process enters round* 1*. If no process aborts in a run of the algorithm, then there must be a successful round in the run.*

Proof. Suppose, for a contradiction, that there is no successful round in the run. Since some process enters round 1 and round 1 is not successful, by Proposition 13, there must be a process that enters round 2. Repeating the same argument, one can conclude that there must be a process that enters round N. Since round N is not successful either, then there must be some process that aborts at the end of round N. □

Lemma 2 *Suppose there is at least one process that enters round* 1*. There is a process that aborts in a run of the algorithm if and only if there is no successful round in the run.*

Proof. Immediate from Propositions 12 and 14. □

Definition 2 *Let* c *be a process called the coordinator, and other processes are called participants. Let* p *be one of the participants. Define algorithm* A *with* c *as the coordinator and* p *as the starting process as follows. The algorithm starts with process* p *sending a* SKIP *message to the coordinator* c*. When* c *receives the* SKIP *message from* p*,* c *sends a* NEWROUND *message to every participant. When a participant receives the* NEWROUND *message from* c*, it sends an* ESTIMATE *message to* c*. After* c *sends out the* NEWROUND *messages for* TO_{NR} *time units,* c *sends a* NEWESTIMATE *message to every participant. When a participant receives the* NEWESTIMATE *message from* c*, it sends an* ACK *message to* c*. After* c *sends out the* NEWESTIMATE *messages for* TO_{NE} *time units,* c *sends a* DECIDE *message to every participant.*

Definition 3 *Let a* benchmark run *of algorithm* A *be a run in which all processes are correct, and it satisfies the following conditions for every participant* q*: (a) the time elapsed from* p *sending the* SKIP *message to* q *receiving the* NEWROUND *message from* c *is at most* TO_S*; (b) the time elapsed from* c *sending the* NEWROUND *message to* c *receiving the* ESTIMATE *message from* q *is at most* TO_{NR}*; (c) the time elapsed from* q *sending the* ESTIMATE *message to* q *receiving the* NEWESTIMATE *message from* c *is at most* TO_E*; (d) the time elapsed from* c *sending the* NEWESTIMATE *message to* c *receiving the* ACK *message from* q *is at most* TO_{NE}*; (e) the time elapsed from* q *sending the* ACK *message to* q *receiving the* DECIDE *message from* c *is at* *most TO_A (see Figure 2(b)).*

Since there are now two algorithms, the following convention is used to reduce possible confusion when referring to runs of an algorithm: the runs of algorithm A are always referred to explicitly as "runs of algorithm A", while the runs of the abortable consensus algorithm in Figure 1 are simply referred to as "runs".

To study the probability of a successful round, one needs to assume that a correct majority of processes (processes that form a majority and are correct) have proposed so that there are enough processes participating in the algorithm to make progress. The following definition is used to address the behaviors of the propose actions.

Definition 7 *A* proposal pattern P *describes when a process proposes in a run. Formally, proposal pattern P is a function from Π to $[0, \infty)$. For each process p, $P(p)$ denotes the time at which process p proposes in this proposal pattern.*

The following lemma is the key element to the entire analysis, because it masks many complicated behaviors, such as process crashes, message losses and message delays that could occur in a successful round, and reduces all different possible cases of successful rounds into a simple and tractable type of benchmark runs.

Lemma 15 *Let r be a round number in $\{1, 2, \dots, N\}$. Let F be a failure pattern in which a majority of processes are correct and the coordinator of round r is also correct. Given a time t, let P be a proposal pattern in which a correct majority of processes in F as well as the coordinator of round r have proposed by time t. Let Σ be the set of all runs with failure pattern F and proposal pattern P such that (a) the first process that enters round r does so after time t, and (b) none of the rounds $1, 2, \dots, r-1$ are successful. The probability that a run in Σ has a successful round r is at least β.*

Proof. For any run $\sigma \in \Sigma$, since there are correct processes in run σ, by Proposition 13, it is easy to see that some process enters round r in σ. First, let Σ_1 be the set of runs in Σ such that all processes entering round r crash right away before sending out any messages. By definition, the round r of every run in Σ_1 is successful. Let $\Sigma_2 = \Sigma \setminus \Sigma_1$. Every run in Σ_2 has some process entering round r and sending out at least one message in round r. Since the probability of a run in Σ_1 having a successful round r is 1, to show the lemma it is sufficient to show that the probability of a run in Σ_2 having a successful round r is at least β.

Let σ be a run in Σ_2. Let p be the first process that sends a round r message in σ. and let t_0 be the time at which p sends the first round r message. By the assumption of the lemma, at least a majority of processes as well as the coordinator of round r already propose by time t_0, which means that these processes participate in the abortable consensus algorithm and are able to response immediately to messages received. Let σ_{t_0} be the partial run of σ from time 0 to time t_0. There could be other runs in Σ_2 that have the same partial run σ_{t_0}. The rest of this proof is to show the following claim:

Claim 1: Given any partial run σ_{t_0}, the probability that the full run extended from σ_{t_0} has a successful round r is at least β.

Once the above claim is proven, the lemma follows immediately since the claim holds for any partial run σ_{t_0}.

Let $\Sigma_3 \subseteq \Sigma_2$ be the set of full runs extended from σ_{t_0}. Let c be the coordinator of round r. Consider the following two cases of σ_{t_0}.

- *Case 1: $p \neq c$, i.e. p is a participant of round r.* The proof is carried out in three steps. The first step is to show that given any benchmark run ρ of algorithm A with c as the coordinator and p as the starting process, one can construct a full run σ extended from σ_{t_0} by following a set of rules that match message delays of certain messages in σ to the delays of messages in ρ. Let $\Sigma_4 \subseteq \Sigma_3$ be the set of full runs constructed following these rules. The second step is to show that the probability that a full run extended from σ_{t_0} is one of the constructed runs using the rules given in step 1 is at least $\beta(p, c) \geq \beta$, i.e., $\Pr(\sigma \in \Sigma_4 \mid \sigma \in \Sigma_3) \geq \beta(p, c)$. The third and final step is to show that every run in Σ_4 has a successful round r.

  Let ρ be a benchmark run of algorithm A with c as the coordinator and p as the starting process. Note that in ρ, (a) coordinator c sends out three sets of messages to all participants: first $n-1$ NEWROUND messages, next $n-1$ NEWESTIMATE messages, and finally $n-1$ DECIDE messages; (b) participant p sends three messages to c: first the SKIP message, then the ESTIMATE message, and finally the ACK messages; (c) any other participant q sends two messages to c: first the ESTIMATE message and then the ACK message. Based on the delay behaviors of these messages, the full run $\sigma \in \Sigma_4$ extended from σ_{t_0} is constructed according to the following rules.

Rule 1 If p sends a SKIP message of round r to c, set the delay of this message to the delay of the SKIP message in ρ.

Rule 2 For any message of round r sent from q_1 to q_2 with the tag being NEWROUND, NEWESTIMATE,

ESTIMATE, or ACK, set the delay of the message to be the delay of the message from q_1 to q_2 with the corresponding tag in ρ. Note that one of q_1 and q_2 must be the coordinator c, and thus every such round r message has a corresponding message in ρ.

Rule 3 If c already decides before entering round r, then when c receives the first round r message, c sends DECIDE messages to all processes in $\Pi \setminus \{c\}$ (line 48). For each process $q \in \Pi \setminus \{c\}$, set the delay of the above DECIDE message from c to q to be the delay of the NEWROUND message from c to q in ρ.

Rule 4 If c decides after sending out NEWROUND messages of round r but before entering Phase NEWESTIMATE, then c sends DECIDE messages to all processes in $\Pi \setminus \{c\}$ right after making the decision (line 23). For each process $q \in \Pi \setminus \{c\}$, set the delay of the above DECIDE message from c to q to be the delay of the NEWESTIMATE message from c to q in ρ.

Rule 5 If c decides in Phase NEWESTIMATE after sending out NEWESTIMATE messages of round r, then c sends DECIDE messages to all processes in $\Pi \setminus \{c\}$ right after making the decision (line 23). For each process $q \in \Pi \setminus \{c\}$, set the delay of the above DECIDE message from c to q to be the delay of the DECIDE message from c to q in ρ.

Rule 6 If a process q different from c receives a NEWROUND message of round r from c, and q already decides by this time, q sends a DECIDE message to c (line 50). Set the delay of this DECIDE message to the delay of the ESTIMATE message from q to c in ρ.

Rule 7 If a process q different from c receives a NEWESTIMATE message of round r from c, and q already decides by this time, q sends a DECIDE message to c (line 50). Set the delay of this DECIDE message to the delay of the ACK message from q to c in ρ.

Rule 8 Any other message sent in the extension of σ_{t_0} can assume any message delay and message loss behavior allowed by the probabilistic distribution.

The above construction is possible because each message transmitted through the same link ℓ has the same message loss probability $p_L(\ell)$ and the same message delay $D(\ell)$, and message delay and loss behaviors are independent. The above construction rules define the set Σ_4.

Claim 2: The probability that a full run extended from σ_{t_0} is one of the constructed runs following the *above rules is at least $\beta(p, c)$, i.e.,* $\Pr(\sigma \in \Sigma_4 \mid \sigma \in \Sigma_3) \geq \beta(p, c)$.

The construction rules only restrict the delays of certain messages in the full run σ. Moreover, it is easy to verify that following the above rules, every message in the benchmark run ρ is used at most once to set the delay of some message in the full run σ. Some messages may not be used at all because a process may already crash or already decide. Thus, the probability that the full run σ is constructed following the above rules is the probability that those restricted messages follow the delay behaviors of the corresponding messages in some benchmark run of algorithm A. The latter probability is no less than the probability that the message delays in a run of algorithm A result in a benchmark run, which is $\beta(p, c)$. Therefore, Claim 2 holds.

Claim 3: Every run in Σ_4 has a successful round r.

Suppose, for a contradiction, that there exists a full run $\sigma \in \Sigma_4$ in which the round r is not successful. From the structure of the abortable consensus algorithm, it is easy to see that there must exist some process in σ that enters round r but times out in one of the phases in round r. Let q be the first process that times out in one of the phases in round r. Thus before q times out in round r, there is no other process that already enters a higher round. The following case study, however, shows that it is impossible for q to time out in any of the phases.

– *Case 1.1: q times out in Phase* SKIP *(line 30).* Let t_1 be the time at which q sends a SKIP message of round r to c in the full run σ (line 26). Since p is the first process that sends a SKIP message of round r to c and p does so at time t_0, it follows that $t_1 \geq t_0$. Let the delay of the SKIP message from p to c be d_1. By Rule 1, d_1 is also the delay of the SKIP message from p to c in ρ. Since c is correct, c receives the SKIP message from p at time $t_0 + d_1$ in σ. If c enters round r and sends the NEWROUND messages of round r to all participants (line 6), let t_2 be the time at which c sends the NEWROUND messages. If c already decides before entering round r, let t_2 be the time at which c receives the first round r message and sends back the DECIDE messages to all participants (line 48). In either case, $t_2 \leq t_0 + d_1$ (because c already proposes by time t_0 so c will respond to the SKIP message received from p at time $t_0 + d_1$ immediately, if c has not already sent out a NEWROUND or a DECIDE message). Let the delay of the message c sends to q at time t_2 be d_2. By Rule 2 or Rule 3, d_2 is also the delay of the NEWROUND message from c to q in ρ. Thus, by condition (a) in the definition of the benchmark run, $d_1 + d_2 \le TO_S$. In run σ, the time elapsed from q sending the SKIP message to the time when the NEWROUND or DECIDE message from c arrives at q is $t_2 + d_2 - t_1 \le t_0 + d_1 + d_2 - t_0 = d_1 + d_2 \le TO_S$. Therefore, q has not timed out yet in Phase SKIP when q receives the NEWROUND or DECIDE message from c. In either case, q leaves Phase SKIP without timing out in the phase — a contradiction.

– *Case 1.2: q times out in Phase* NEWROUND *(line 9).* Thus $q = c$. Let t_1 be the time at which c sends out NEWROUND messages to all participants in σ (line 6). Thus $t_1 \ge t_0$. Since there are a correct majority of processes in the run σ, at least $\lfloor n/2 \rfloor$ participants are correct. Moreover, by the assumption of the lemma, at least $\lfloor n/2 \rfloor$ correct participants has proposed by time t_0, so they will response to round-r messages from c immediately. Thus, there must exist a correct participant q' such that q' has proposed by time t_0 but c does not receive an ESTIMATE or a DECIDE message from q' by the time $t_1 + TO_{NR}$; otherwise, c would have either decided or gathered $\lfloor n/2 \rfloor$ ESTIMATE messages and moved to Phase NEWESTIMATE. Let d_1 be the delay of the NEWROUND message of round r from c to q'. By Rule 2, d_1 is also the delay of the NEWROUND message from c to q' in ρ. By condition (b) in the definition of the benchmark run, $d_1 \le TO_{NR}$. So by time $t_1 + d_1$ at which q' receives NEWROUND message of round r from c, c has not timed out yet. Since c is the first process to time out in round r, q' cannot have passed Phase SKIP of round r by time $t_1 + d_1$. Thus, there are only two possibilities for q': either q' has decided by time $t_1 + d_1$, in which case q' replies a DECIDE message to c (line 50) at time $t_1 + d_1$, or q' enters the Phase ESTIMATE of round r and sends an ESTIMATE message to c (line 33) at time $t_1 + d_1$. Let d_2 be the delay of the message q' sends to c at time $t_1 + d_1$. By Rule 2 or Rule 6, d_2 is also the delay of the ESTIMATE message from q' to c in ρ. By condition (b) in the definition of the benchmark run, $d_1 + d_2 \le TO_{NR}$. Therefore, c receives a ESTIMATE or DECIDE message from q' at time $t_1 + d_1 + d_2 \le t_1 + TO_{NR}$ — contradicting to the definition of q'.

– *Case 1.3: q times out in Phase* ESTIMATE *(line 36).* Let t_1 be the time at which q sends the ESTIMATE message of round r to c in σ (line 33). According to the abortable consensus algorithm, it is easy to see that if q sends the ESTIMATE message of round r to c at time t_1, then q receives the NEWROUND message of round r from c at time t_1. Let d_1 be the delay of the NEWROUND message from c to q, so c sends the NEWROUND messages at time $t_1 - d_1$. By Rule 2, d_1 is also the delay of the NEWROUND message from c to q in ρ. Let d_2 be the delay of the NEWESTIMATE message from c to q in ρ. By condition (c) in the definition of the benchmark run, and the fact that in ρ, c sends the NEWESTIMATE messages exactly TO_{NR} time units after sending the NEWROUND messages, $TO_{NR} + d_2 - d_1 \leq TO_E$. This implies that c cannot time out in Phase NEWROUND or receive a higher round message and skip the rest of round r, because if so, c or some other process would have timed out in round r before the time $t_1 - d_1 + TO_{NR} \leq t_1 + TO_E$ — contradicting to the assumption that q is the first to time out in round r and q times out at time $t_1 + TO_E$. Thus, c either collects enough ESTIMATE messages and moves to Phase NEWESTIMATE, or c decides in Phase NEWROUND. In the first case, let t_2 be the time at which c sends the NEWESTIMATE messages of round r to the participants (line 16); in the second case, let t_2 be the time at which c sends the DECIDE messages to the participants right after the decision (line 23). In either case, $t_2 \leq t_1 - d_1 + TO_{NR}$. By Rule 2 or 4, the delay of the message that c sends to q at time t_2 is d_2. Thus, the time elapsed from q sending the ESTIMATE message to the time at which either a NEWESTIMATE or a DECIDE message arrives at q is $t_2 + d_2 - t_1 \leq t_1 - d_1 + TO_{NR} + d_2 - t_1 \leq TO_E$. That is, q receives a NEWESTIMATE message of round r or a DECIDE message from c before q times out in Phase ESTIMATE. In either case, q leaves Phase ESTIMATE without timing out in the phase — a contradiction.

- *Case 1.4: q times out in Phase* NEWESTIMATE *(line 19).* The argument in this case is very similar to that of case 1.2, with Rule 2 and 7 and condition (d) in the definition of the benchmark run being used in this case.
- *Case 1.5: q times out in Phase* ACK *(line 43).* The argument in this case is very similar to that of Case 1.3, with Rule 5 and condition (e) in the definition of the benchmark run being used in this case.

• *Case 2: $p = c$, i.e. p is the coordinator of round r.* The proof in this case is very similar to that of Case 1. In this case, one can pick any participant p' and base the argument on the benchmark runs with c as the coordinator and p' as the starting process. When constructing the full run σ from a benchmark run ρ, the construction rules of Case 1 still apply, with the exception that Rule 1 can be omitted and Rule 3 is void since c enters round r. Claims similar to Claims 2 and 3 of Case 1 can also be shown with the same structure of the argument.

Claim 1 is proven after both Case 1 and 2 are shown, and thus the lemma holds. $\square$

It is easy to see that given a reasonable set of timeout values, the benchmark probability β should be greater than zero. The rest of this section will show that a non-zero β is sufficient to achieve α-Abortability, for any arbitrarily small α.

The following lemma applies Lemma 15 and shows that after a majority of processes propose and assuming that no process has aborted by that time, the probability of consecutive unsuccessful rounds declines exponentially.

Lemma 16 *Let F be a failure pattern in which a majority of processes are correct. For some given $j, k \in \{1, 2, 3, \ldots\}$, let $N = j + kn$. Given a time t, let P be a proposal pattern in which a correct majority of processes in F have proposed by time t. Let Σ be the set of runs with failure pattern F and proposal pattern P. If in any of the runs in Σ, no process has entered a round higher than j by time t, then the probability that a run in Σ has none of the rounds $1, 2, \ldots, j + kn$ being successful is at most $(1 - \beta)^{k\lceil (n+1)/2 \rceil}$.*

Proof. For $i \in \{1, 2, \ldots, j + kn\}$, let p_i be the probability that a run in Σ does not have a successful round i given that the run does not have successful rounds $1, 2, \ldots, i - 1$. Thus, the probability that a run in Σ has none of the rounds $1, 2, \ldots, j + kn$ being successful is $\prod_{i=1}^{j+kn} p_i$.

Let Γ be the set of the correct processes that have proposed by time t according to the failure pattern F and the proposal pattern P. Thus $|\Gamma| \geq \lceil (n + 1)/2 \rceil$. For a round $i > j$, if the coordinator of round i is in Γ, then Lemma 15 is applicable to round i, which means $p_i \leq 1 - \beta$. Since the coordinator rotates through all processes, for any n consecutive rounds, at least $\lceil (n + 1)/2 \rceil$ rounds are such that the coordinators of these rounds are in Γ. Thus, for at least $k\lceil (n + 1)/2 \rceil$ rounds from round $j + 1$ to round $j + kn$, the corresponding probability p_i is at most $1 - \beta$. Therefore, $\prod_{i=1}^{j+kn} p_i \leq (1 - \beta)^{k\lceil (n+1)/2 \rceil}$. $\square$

Definition 5 *Suppose that a majority of processes are correct. Let* proposal delay *V be a random variable representing the elapsed time from the time when the first process proposes to the time by which a correct majority of processes (processes that form a majority and are correct) have proposed. The probability that V is within a given value t, denoted by $\Pr(V \leq t)$, tends to 1 when t tends to ∞.*

Proposition 17 *Suppose p is the first process that enters a round r, and it enters round r at time t. If a process enters round $r+1$, then it must enter round $r+1$ no sooner than $t+T_m$, where $T_m = \min(TO_{NR}, TO_{NE}, TO_S, TO_E, TO_A)$.*

Proof. Let q be the first process that enters round $r+1$. It must enter round $r+1$ by timing out in one of the phases in round r. Thus it must have spent at least one entire phase in round r. Therefore, the time at which q enters round $r+1$ must be at least T_m time units after the first process enters round r. □

Lemma 18 (α-Abortability) *Suppose the benchmark probability $\beta > 0$. Let $N = j + kn$ for some $j, k \in \{1, 2, 3, \ldots\}$. There exists $\alpha = \gamma + (1-\gamma)(1-\beta)^{k\lceil (n+1)/2 \rceil} < 1$ where $\gamma = \Pr(V > jT_m)$, V is the proposal delay, and $T_m = \min(TO_{NR}, TO_{NE}, TO_S, TO_E, TO_A)$, such that for any failure pattern in which a majority of processes are correct, the probability that there exists some process that aborts in a run with the failure pattern is at most α.*

Proof. Let F be a failure pattern in which a majority of processes are correct. Let Σ be the set of runs with failure pattern F. Let $\Sigma_1 \subseteq \Sigma$ be the set of runs in which some process aborts. Then $\Pr(\sigma \in \Sigma_1 | \sigma \in \Sigma)$ is the probability that some process aborts in the failure pattern F. Let $\Sigma_2 \subseteq \Sigma$ be the set of runs such that within jT_m time units after the first process proposes, a majority of processes have proposed. By the definition of proposal delay V, $\Pr(\sigma \in \Sigma_2 | \sigma \in \Sigma) = \Pr(V \leq jT_m) = 1 - \gamma$. Thus, one has

$$\begin{aligned}
\Pr(\sigma \in \Sigma_1 | \sigma \in \Sigma) &= \Pr(\sigma \in \Sigma_1 \cap \Sigma_2 | \sigma \in \Sigma) + \Pr(\sigma \in \Sigma_1 \cap (\Sigma \setminus \Sigma_2) | \sigma \in \Sigma) \\
&\leq \Pr(\sigma \in \Sigma_1 | \sigma \in \Sigma_2) \cdot \Pr(\sigma \in \Sigma_2 | \sigma \in \Sigma) + \Pr(\sigma \in \Sigma \setminus \Sigma_2 | \sigma \in \Sigma) \\
&= \Pr(\sigma \in \Sigma_1 | \sigma \in \Sigma_2)(1-\gamma) + \gamma.
\end{aligned}$$

The term $\Pr(\sigma \in \Sigma_1 | \sigma \in \Sigma_2)$ denotes the probability that some process aborts given that a correct majority of processes have proposed within jT_m time units after the first process proposes. By Lemma 2, a process aborting in a run is equivalent to no rounds from 1 to $j + kn$ of the run being successful. Let P be a proposal pattern in which a correct majority of processes in F have proposed within jT_m time units after the first process proposes, and let t_0 be the time at which the first process proposes in P. Let Σ_2^P be the set of runs in Σ_2 with proposal pattern P. By Proposition 17, each round lasts at least T_m time units. Thus no process has entered a round higher than j by time $t_0 + jT_m$. Therefore, Lemma 16 is applicable to Σ_2^P, which means the probability that some process aborts in Σ_2^P is at most $(1-\beta)^{k\lceil (n+1)/2 \rceil}$. Since the above holds for every proposal pattern P in which a correct majority of processes in F have proposed within jT_m time units after the first process proposes, the probability that some process aborts in Σ_2 is at most $(1-\beta)^{k\lceil (n+1)/2\rceil}$, i.e. $\Pr(\sigma \in \Sigma_1 | \sigma \in \Sigma_2) \leq (1-\beta)^{k\lceil (n+1)/2\rceil}$. This completes the proof to the inequality $\Pr(\sigma \in \Sigma_1 | \sigma \in \Sigma) \leq \gamma + (1-\gamma)(1-\beta)^{k\lceil (n+1)/2\rceil}$. Finally, when $\beta > 0$, it is obvious that the above probability is less than 1. Therefore the lemma holds. $\square$

Theorem 1 *The algorithm given in Figure 1 satisfies Uniform Validity, Uniform Agreement, and Termination properties. Furthermore, if the benchmark probability β is nonzero, the algorithm with the abort round threshold $N = j+kn$, $j, k \in \{1, 2, 3, \ldots\}$, satisfies α-Abortability with $\alpha = \gamma+(1-\gamma)(1-\beta)^{k\lceil (n+1)/2\rceil} < 1$, where $\gamma = \Pr(V > jT_m)$, V is the proposal delay, and $T_m = \min(TO_{NR}, TO_{NE}, TO_S, TO_E, TO_A)$. Therefore, the algorithm implements abortable uniform consensus with α tending to zero as N tends to infinity.*

Proof. The theorem follows from Lemmata 4, 5, 7, and 18. $\square$

B Proof of Theorem 2

Definition 6 *Given a timeout value TO, for any link ℓ, let $\lambda(\ell)$ be the probability that a message sent on link ℓ is delivered within TO time units after it is sent. Let* delivery probability *λ of a network for the timeout TO be the minimum of $\lambda(\ell)$'s for all links in the network.*

Lemma 3 *Suppose the delivery probability of a network for a given timeout TO is $\lambda > 0$. Set the timeout parameters of the abortable consensus algorithm as follows: $TO_{NR} = TO_{NE} = TO_S = 2TO$, and $TO_E = TO_A = 3TO$. Then the benchmark probability β has a lower bound λ^{5n-4}.*

Proof. First, when all processes are correct, every run of algorithm A always has $5n-4$ messages: one SKIP message, and $n-1$ NEWROUND, ESTIMATE, NEWESTIMATE, ACK, and DECIDE messages each.

Claim: If each of the above $5n-4$ message is delivered within TO time units, then the resulting run is a benchmark run.

In fact, the total delay of the SKIP message and any one of the NEWROUND message is at most $2TO = TO_S$. Since the coordinator c always sends the NEWROUND messages immediately when it receives the SKIP message from p, condition (a) in the definition of the benchmark run is satisfied. For any process q, the total delay of the NEWROUND message to q and the ESTIMATE message from q to c is at most $2TO = TO_{NR}$, and q sends the ESTIMATE message immediately when it receives the NEWROUND message, so condition (b) of the benchmark run is satisfied. For any process q, q cannot send out the ESTIMATE message earlier than the time c sends out the NEWROUND messages, and q receives the NEWESTIMATE message no later than TO time units after c sends the NEWESTIMATE messages. Since c sends the NEWESTIMATE messages exactly TO_{NR} time units after it sends out NEWROUND messages, the time elapsed from q sending the ESTIMATE message to q receiving the NEWESTIMATE message is at most $TO_{NR} + TO = 3TO = TO_E$ time units. Thus, condition (c) of the benchmark run is satisfied. Condition (d) and (e) can be similarly verified. Therefore, the claim holds.

The claim implies that the benchmark probability β is bounded from below by the probability that each of the $5n-4$ messages is delivered within TO time units. By the definition of delivery probability λ, the probability that each of the $5n-4$ messages is delivered within TO time units is at least λ^{5n-4}. The lemma holds. □

Theorem 2 *Suppose the delivery probability of a network for a given timeout TO is $\lambda > 0$. For any value $\alpha \in (0,1)$, suppose $j \in \{1,2,3,\dots\}$ is such that $\Pr(V > 2jTO) \leq \alpha/2$, where V is the proposal delay. Then the algorithm in Figure 1 satisfies α-Abortability for the given α if the parameters of the algorithm are set up as follows: $TO_{NR} = TO_{NE} = TO_S = 2TO$, $TO_E = TO_A = 3TO$, and $N = j + kn$ where*

$$k = \left\lceil \frac{\log(\alpha/2)}{\lceil (n+1)/2 \rceil \log(1-\lambda^{5n-4})} \right\rceil .$$

Proof. By Theorem 1, the algorithm with $N = j + kn$ has an upper bound on the abort probability as $\alpha_0 = \gamma + (1-\gamma)(1-\beta)^{k\lceil (n+1)/2\rceil}$. With the timeout settings as given in the theorem statement, $T_m = \min(TO_{NR}, TO_{NE}, TO_S, TO_E, TO_A) = 2TO$, and thus $\gamma = \Pr(V > jT_m) = \Pr(V > 2jTO) \leq \alpha/2$. From the k given in the theorem statement,

$$k \geq \frac{\log(\alpha/2)}{\lceil (n+1)/2 \rceil \log(1-\lambda^{5n-4})},$$

and thus,

$$(1-\lambda^{5n-4})^{k\lceil (n+1)/2\rceil} \leq \alpha/2$$

By Lemma 3, $\beta \geq \lambda^{5n-4}$. Thus,

$$\begin{aligned} \alpha_0 &= \gamma + (1-\gamma)(1-\beta)^{k\lceil (n+1)/2\rceil} \\ \alpha_0 &\leq \gamma + (1-\beta)^{k\lceil (n+1)/2\rceil} \end{aligned}$$

$$\begin{aligned}\alpha_0 &\leq \alpha/2 + (1-\lambda^{5n-4})^{k\lceil (n+1)/2\rceil} \\ &\leq \alpha/2 + \alpha/2 = \alpha.\end{aligned}$$

Therefore, the algorithm satisfies the α-Abortability for the given α. □

C Proof of Theorem 3

This appendix proves the correctness of the PRBCast algorithm given in Figure 3.

Theorem 3 *The algorithm in Figure 3 implements probabilistic reliable broadcast in the probabilistic network model, with the probability thresholds $\gamma_v = 1$ and $\gamma_a = 1 - p_L^k$, where p_L is the maximum of message loss probability $p_L(\ell)$'s for all communication links ℓ in the system, and k is the maximum number of repeated send actions on a message as described in the algorithm.*

Proof. Integrity property is obviously satisfied. Validity is also satisfied with $\gamma_v = 1$, since a correct process p always PR-delivers m if p PR-broadcasts m. For the Agreement property, if p and q are correct and p PR-delivers m, then p must have PR-sent m to q. Since p and q are correct and p will send m to q k times if not receiving q's acknowledgment, the probability that q does not receive m at all is the probability that all these k messages are lost, which is $p_L(\ell)^k$ where ℓ is the link from p to q. So the probability that q receives m from p is at least $1 - p_L(\ell)^k \geq 1 - p_L^k$. If q receives m, then q PR-receives m and thus q PR-delivers m. Therefore, the threshold for Agreement $\gamma_a = 1 - p_L^k$. □

D Proof of Theorem 4

This appendix proves the correctness of the PABCast algorithm given in Figure 4.

Lemma 19 (Integrity) *Every message is delivered at most once at each process, and only if it was previously broadcast.*

Proof. Trivial. □

Lemma 20 *If a correct process p executes the abortable consensus for an infinite number of times and has an infinite number of decide outputs from these executions of the abortable consensus, then a majority of processes must be correct.*

Proof. Suppose, for a contradiction, that a majority of processes are faulty in the system. Let Q be the set of faulty processes. Let t be the time when the last faulty process crashes. At least one execution of the abortable consensus is started after time t in which p decides a value.[6] Suppose p decides v_0 in this consensus. Let r_0 denote this run of consensus.

Consider a different run r_1 in which all processes in Q are correct and all other processes crashes at time 0, and all processes in Q propose $v_1 \neq v_0$. Since Q has a majority of processes, by the α-Abortability of the abortable consensus, the probability that some process in Q aborts is at most $\alpha < 1$. Thus, there exists a run in which all processes in Q decides, and by the Uniform Validity property, the decision value has to be v_1.

Now consider run r_2 combined from run r_0 and r_1, in which all processes are correct, but initially, all messages sent between processes in Q and $\Pi \setminus Q$ are delayed for a long time. Processes in Q behave just like in run r_1, in which case they decide on value v_1. Processes not in Q behave like in run r_0, in which case p decides on value $v_0 \neq v_1$. With the probabilistic network model, this run may occur even though the probability is small. However, in this run, correct processes decide on different values, violating Agreement property of the abortable consensus. $\square$

Let γ_a^R and γ_v^R be the probability thresholds for the Probabilistic Agreement and Probabilistic Validity properties of PRBCast, respectively.

Lemma 21 (Probabilistic Agreement) *There exists a $\gamma_a^A = \gamma_a^R(1-\alpha) > 0$ such that for any failure pattern in which processes p and q are correct, if p PA-delivers message m, then the probability that q PA-delivers m is at least γ_a^A.*

Proof. If p PA-delivers m, then p PR-delivers m. By the Probabilistic Agreement of PRBCast, q PR-delivers m with probability at least γ_a^R. Consider the case where q PR-delivers m but does not PA-deliver m. Then q executes an infinite number of the abortable consensus with m in its proposals. In none of these executions q has the abort output, because otherwise, q would PA-deliver its own proposal, which includes m. Thus, q decides an infinite number of times. By Lemma 20, a majority of processes are correct in the run. Let k be the index of the abortable consensus execution at the end of which p PA-delivers m. In this execution, at least one of p and q aborts, because otherwise they would decide on the same value, which implies that q will PA-deliver m. Thus there is a correct process that aborts in the k-th abortable consensus execution when a majority of processes are correct. By the α-Abortability of the abortable consensus, the probability that this occurs is at most α. Therefore, the probability that q PA-delivers m given that p PA-delivers m is at least

[6] Implicitly we require that there be only a finite number of consensus executions during any finite time period. This is a very realistic requirement for all practical purposes.

$\gamma_a^R(1-\alpha)$. □

Lemma 22 *If a correct process p PR-delivers message m, then the probability that p PA-delivers m is at least $\gamma_a^R(1-\alpha)$.*

Proof. If p PR-delivers m, then $m \in R\_delivered_p$. Suppose p does not PA-deliver m. We need to analyze the probability of this case. In this case, $m \notin A\_delivered_p$. There exists a number K_0 such that for all $k \geq K_0$, process p calls propose$(k, A_undelivered_p)$ with $m \in A\_undelivered_p$. By the Termination property of abortable consensus, each propose call returns with either decide or abort. None of the calls could return abort, because otherwise p would PA-deliver all messages in $A\_undelivered_p$, including m. So for all $k \geq K_0$, p has decided in the k-th abortable consensus run, but none of the decision values in these consensus runs is its own proposal. By Lemma 20, a majority of processes are correct in this run.

Since a faulty process can only make a finite number of proposals, there exists a number K_1 such that for all $k \geq K_1$, all the decision values are from correct processes. Thus, there is at least one correct process $q \neq p$ such that an infinite number of decision values for consensus rounds $k \geq K_1$ are proposed by q. None of these proposals include message m. There are two possible cases here. First, q never PR-delivers m. By the Probabilistic Agreement property of PRBCast, this case could happen with probability at most $1 - \gamma_a^R$. Second, q PR-delivers m. In this case, q must PA-deliver m at some point. Otherwise, after some K_2, every proposal made by q will include m, contradicting to the fact that p decides an infinite number of times on q's proposals that do not include m. Let k be the index of the abortable consensus execution at the end of which q PA-delivers m. In this execution, either q aborts the consensus run and decides on its local proposal, or q decides a value including m but p aborts the consensus run. Since we know that a majority of processes are correct, by the α-Abortability of the abortable consensus, the probability that either p or q aborts the k-th abortable consensus execution is at most α.

Therefore, the probability that p PA-delivers m is the probability that neither of the above two cases occur, which is at least $\gamma_a^R(1-\alpha)$. □

Lemma 23 (Probabilistic Validity) *There exists an $\gamma_v^A = \gamma_v^R \gamma_a^R (1-\alpha) > 0$ such that for any failure pattern in which process p is correct, if p PA-broadcasts message m, then the probability that p PA-delivers m is at least γ_v^A.*

Proof. If p PA-broadcasts m, then p PR-broadcasts m. By the Probabilistic Validity of PRBCast, p PR-delivers m with probability at least γ_v^R. By Lemma 22, if p PR-delivers m, then p PA-delivers m with probability at least $\gamma_a^R(1-\alpha)$. Therefore, the probability that p PA-delivers m given that p PA-broadcasts m is at least $\gamma_v^R\gamma_a^R(1-\alpha)$. □

Lemma 24 (Probabilistic Order) *Suppose a majority of processes are correct. There exists a $\gamma_o^A = (1-\alpha)^2 > 0$ such that for any failure pattern in which processes p and q are correct, if p and q both PA-deliver m and m', then the probability that they do so in the same order is at least γ_o^A.*

Proof. Suppose p PA-delivers m and m' at the end of the k_p-th and the k'_p-th abortable consensus executions, and q PA-delivers m and m' at the end of the k_q-th and the k'_q-th abortable consensus executions. Let $k = \min(k_p, k_q)$ and $k' = \min(k'_p, k'_q)$. So p and q have not PA-delivered m before the k-th consensus, and they have not PA-delivered m' before the k'-th consensus. If both p and q decide in both the k-th and the k'-th of the abortable consensus executions, by the Agreement property of the abortable consensus, they decide on the same set of values $msgSet^k$ and $msgSet^{k'}$, respectively, with $m \in msgSet^k$ and $m' \in msgSet^{k'}$. Since p and q have not PA-delivered m and m' before the k-th and the k'-th consensus, respectively, they PA-deliver m and m' in the same order. The probability of this case is at least the probability that no process aborts in the k-th or the k'-th consensus. Since a majority of processes are correct, by the α-Abortability of abortable consensus, the probability that p and q PA-deliver m and m' in the same order is at least $(1-\alpha)^2$. □

Theorem 4 *Suppose the probabilistic reliable broadcast algorithm used in Figure 4 has γ_a^R and γ_v^R as the thresholds for the Probabilistic Agreement and Probabilistic Validity properties, respectively. Suppose the abortable consensus algorithm used in Figure 4 has α as the threshold for the α-Abortability property. The PABCast algorithm given in Figure 4 satisfies the Integrity property, the Probabilistic Agreement property with threshold $\gamma_a^A = \gamma_a^R(1-\alpha)$, the Probabilistic Validity property with threshold $\gamma_v^A = \gamma_v^R\gamma_a^R(1-\alpha)$. If a majority of processes are correct, it also satisfies the Probabilistic Order property with threshold $\gamma_o^A = (1-\alpha)^2$. Therefore, the algorithm implements probabilistic atomic broadcast when a majority of processes are correct.*

Proof. Direct from Lemmata 19, 21, 23, and 24. □