

On the Probabilistic Omission Adversary

Taisuke Izumi and Koichi Wada

Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya, Aichi 466-8555, Japan
{t-izumi, wada}@nitech.ac.jp

Abstract. This paper newly proposes a novel round-based synchronous system suffering crash and probabilistic omission failures. In this model, a novel class of adversaries, called *p-probabilistic omission adversary* (*p-POA*) is introduced. In addition to the ability of complete control of the crash-failure behavior, *p-POA* can select any subset of all transmitted messages as *omission candidates*. Then, each message in the omission candidates is lost with probability p . This paper investigates the feasibility and complexity of the consensus problem under *p-POA*. We first show two impossibility results that (1) for any $p > 0$, there exists no uniform consensus algorithm tolerating more than or equal to $n/2$ crash failures, and that (2) for any $p > 0$, any uniform consensus algorithm cannot halt. We also show two consensus algorithms CPO and F-CPO. Both algorithms work under $(1/2)$ -POA and respectively have distinct advantages. The algorithm CPO can tolerate at most $n/2 - 1$ crash failures and achieves $O(f)$ expected round complexity, where f is the actual number of crash failures. This implies that CPO has maximum crash-failure resiliency. While the second algorithm F-CPO assumes the maximum number of crash failures less than $n/3$, it achieves $f + O(1)$ round complexity in expectation. Since it is known that the lower bound for crash-tolerant consensus is $f + 1$, this result implies that only a constant number of extra rounds is necessary to tolerate a drastic number of message omissions.

1 Introduction

Consensus problem is one of fundamental and important problems for designing fault-tolerant distributed systems. In the consensus problem, each process proposes a value, and has to agree on a common value that is proposed by a process unless it crashes. The consensus problem has many practical applications, e.g., atomic broadcast [3, 10], shared object [1, 11], weak atomic commitment [9] and so on. However, despite of such applications, it has no deterministic solution in asynchronous systems subject to only a single crash failure [7]. Thus, several consensus algorithms have been considered on systems with some additional assumptions [5, 6, 16, 3]. Especially, the *round-based synchrony* is one of the most commonly used assumptions for designing consensus algorithms [16, 17]. Executions of the round-based synchronous systems proceed along consecutive

communication-closed rounds, where “communication-closed” means that every message sent in a round is received in the same round.

In literatures about the synchronous consensus, the failure behavior other than crash is also considered. The *omission failure* is one of such failures, which allows a part of communication messages to be lost in transmission. Generally, for omission failures, two kinds of modelings have been considered. The most traditional one is the component failure model, which statically specifies the set of faulty components (i.e., faulty edges and/or faulty processes) and only messages related to faulty components can be omitted. Since the component failure model is a natural extension of crash-failure models, it is widely accepted in many studies [2, 4, 8, 14–16]. However, in practical sense, it is not necessarily the best model: In real systems, the omission failures can occur on any component, but their occurrences are not permanent. Thus, the omission failures should be modeled as a *transient* and *ubiquitous* one, which is the contrary of the component failure model. The second model, called *dynamic failure model*, is one that absorbs these features [13, 18]. While the component failure model restricts its failure behavior by specifying faulty components, the dynamic failure model bounds the behavior by the number of failures. More precisely, the dynamic failure model only assumes the existence of upper bound for the number of omission that can occur in one round. This model is more plausible than the component failure model in a certain type of real systems. Nevertheless, both the component and dynamic failure models leave the disadvantage that the failure influence is too strong: Only a small fraction of failures makes a number of uniform tasks unsolvable. For example, in a complete network of n processes, if $2n - 1$ omission failures are possible, the system can suffer the isolation of a non-crashed process, i.e., all messages from/to the process are lost by omission failures. Then, uniform solution of the consensus problem is clearly impossible. This disadvantage derives from the fact that the *adversary*, which is a daemon determining the failure behavior of system executions, can deterministically control the targets where failures appear.

In this paper, as the model to circumvent the above disadvantage, we newly introduce a round-based synchronous system suffering *crash* and *probabilistic omission failures*, and consider consensus algorithms on this system. Specifically, our model introduces the novel class of adversaries, called *p-probabilistic omission adversary* (p -POA). The p -POA is a generalization of the traditional crash-failure adversary, and thus it has all ability that the crash-failure adversary has. In addition, at each round, p -POA can select any subset of all transmitted messages (even the set consisting of all messages is possible) as *omission candidates*. Each message in the omission candidate is lost with probability p , where p is a system parameter of the adversary. Because of its probabilistic nature, the POA does not cause the isolation of a part of processes, and thus it can handle more drastic number of message omissions than existing omission failure models.

This paper investigates the feasibility of the consensus problem under the p -POA. Since communications liveness between any pair of processes is probabilistically guaranteed in our model, it seems not so surprising that the uniform

consensus can be solved in our model (actually, we propose two uniform consensus algorithms under p -POA in this paper). However, it is quite nontrivial how probabilistic omissions affect the complexity of the consensus problem, which is the main interest of our study. In the followings, we summarize the contribution of this paper:

- We show two fundamental impossibility result about the consensus problem under p -POA. The first result is that for any $p > 0$, there is no uniform consensus algorithm that can tolerate more than or equal to $n/2$ crash failures. The second one is that for any $p > 0$, there is no *halting* algorithm to solve the uniform consensus, i.e., in any consensus algorithm under p -POA, each process must continue to execute the algorithm even after it decides.
- We propose a consensus algorithm CPO working correctly under the $(1/2)$ -POA. This algorithm assumes that the maximum number of failures t is less than $n/2$, which implies the maximum failure resiliency. The algorithm has $O(f)$ expected round complexity, where f is the actual number of crash failures. In the sense that the round complexity depends only the actual number of crash failures, this algorithm is *early-deciding*.
- We also propose another early-deciding consensus algorithm F-CPO working correctly under the $(1/2)$ -POA. This algorithm requires the stronger assumption of $t < n/3$ than CPO, but has $f + O(1)$ round complexity in expectation. Interestingly, since it is known that the lower bound for crash-tolerant consensus is $f + 1$, this result implies that only a constant number of extra rounds is necessary to tolerate a drastic number of omission (about a half of all messages).

The paper is organized as follows: In Section 2, we introduce the system model, and the definition of the consensus problem. Section 3 shows two impossibility results. Sections 4 and 5 provides the algoirhm CPO and F-CPO, respectively. Finally we conclude this paper in Section 6.

2 Preliminaries

2.1 Distributed Systems

We consider a distributed system with round-based synchrony. The distributed system consists of n processes $\mathcal{P} = \{p_0, p_1, p_2, \dots, p_{n-1}\}$ that are completely connected by communication links, that is, any pair of processes can communicate with each other by directly exchanging messages. The system is round-based, that is, its execution is a sequence of synchronized *rounds* identified by $0, 1, 2$, etc. Each round r consists of three steps:

Send step Each process p_i sends messages.

Receive step Each process p_i receives all the messages sent to p_i at the send step of the current round.

Local processing step Each process p_i executes local computation.

2.2 Failure Models

An *adversary* is a daemon determining the behavior of failures at each round. Notice that the adversary always makes a decision by worst possible cases. In this paper, to handle both crash and probabilistic omission, we introduce a novel class of adversaries, called *p-Probabilistic Omission Adversaries* (*p*-POA). Same as the traditional crash adversary, it can completely control the behavior of process crashes. If a process p_i is designated as crash during round r , it makes no operation subsequently. Then, the messages sent by p_i at round r may or may not be received, which is also controlled by the adversary. In addition to the ability of crash adversaries, *p*-POA can omit each in-transmission message with probability p , where p is a system-specific parameter. More precisely, it determines *omission candidates* C_r at each round r , which is an arbitrary subset of all messages sent at round r . Every message $m \notin C_r$ is guaranteed to be transmitted correctly. On the other hand, each message $m \in C_r$ is lost with probability p . As a summary of the above model, the execution of one round r in our model can be described as follows:

1. The sending step is executed. Then, for each non-crashed process p_i , the adversary determines whether p_i crashes at the current round or not. In addition, if a process p_i is determined as crash, the adversary also determines its crash behavior (that is, determines which messages that will be sent by p_i at the current round are actually sent).
2. The adversary decides omission candidates C_r . Each message in C_r is lost with probability p .
3. The receiving step is executed. All remaining messages are correctly received.
4. The local processing step is executed.

It should be noted that the above failure model is a weaker model of the traditional crash-failure models: The *p*-probabilistic omission adversary can deterministically control the execution of the system so that it will behave as the traditional crash-failure systems. This implies that any complexity lower bounds for the crash-failure model also hold in our model.

There is an upper bound t on the number of processes that can crash. Throughout this paper, we assume $t < n/2$. This assumption is necessary to exist a consensus algorithm on our model (the necessary proof is given in Section 3). The actual number of crash processes is denoted by f ($\leq t$). A process is said to be *correct* if it never crashes, and a round r is said to be *correct* when no process crashes during the round r .

2.3 Consensus Problem

In a consensus algorithm, each correct process initially proposes a value, and eventually chooses a decision value from the values proposed by processes so that all processes decide the same value. The *uniform consensus* is a stronger variant of the consensus. It disallows faulty processes to disagree on the decided value. The standard specification of uniform consensus is described as follows:

Termination Every correct process eventually decides.

Uniform Agreement No two processes decide different values.

Validity If a process decides a value v , then, v is a value proposed by a process.

Since we consider the probabilistic omission, it is clear that there is no algorithm satisfying the above termination property in our model. For example, the execution where all messages are omitted is a possible execution in our model. In such execution, no algorithm can correctly reach decisions. Thus, we have to relax the specification of the consensus problem such that it allows a probabilistic guarantee of Termination property. In this paper, we define the consensus problem by the uniform agreement property, the validity property, and the following probabilistic termination property:

Probabilistic Termination Every correct process decides with probability 1.

Notice that decision does not necessarily implies the *halt* of the algorithm. Even after a process decides, it may work for helping the decision of other processes.

3 Impossibility Results

In this section, we present two fundamental impossibility results about the consensus problem on the system with the p -probabilistic omission adversary (for lack of space, we omit the proofs).

Theorem 1 For any $p > 0$, there is no algorithm solving the consensus problem under the p -probabilistic omission adversary if $t \geq n/2$ holds.

Theorem 2 For any $p > 0$ and $t > 0$, there is no *halting* consensus algorithm under the p -probabilistic omission adversary.

4 The $O(f)$ -Round Algorithm for $t < n/2$

This section provides a consensus algorithm under the $(1/2)$ -probabilistic omission adversary, which can tolerate less than $n/2$ crash processes. Because of Theorem 1, this algorithm achieves the maximum crash-failure resilience. The basic idea of our algorithm derives from Chandra and Toueg's algorithm using eventually strong failure detectors [3]. Before the presentation of our algorithm, we first discuss about the round complexity of a group communication primitive under the $(1/2)$ -probabilistic omission adversary in the following subsection, which is used to construct our consensus algorithm.

Algorithm 2RB(d): Code for p_i

```

1: variable:
2:    $M_i$  : init  $\perp$ 

3: At round 1:
4:   if  $p_i$  is the source process then
5:      $\text{Send}_i(d)$  to all processes
6:      $M_i \leftarrow d$ 
7:   endif
8: At round 2:
9:   if  $M_i \neq \perp$  then
10:     $\text{Send}_i(d)$  to all processes
11:  endif
12:  if a message  $d$  is received from  $p_j$  then
13:     $M_i \leftarrow d$ 
14:  endif

```

Fig. 1. Algorithm 2RB(d): 2-Round Broadcast Algorithm for Transmission Data d

4.1 The Round Complexity of All-to-all Broadcast

In our adversary model, there is no guarantee that a message sent by a correct process is necessarily received. Thus, if a process wants to broadcast an information d , one-time broadcasts bring only a slight possibility that all processes can correctly receive d . Thus, this subsection investigates how many rounds are sufficient to guarantee that all processes receive the information d with high probability.

First, we introduce an broadcast algorithm called *2-round Broadcast*(2RB). The pseudocode description of the 2RB algorithm for each process p_i is given in Figure 1. The variable d is the broadcast information, and the local variable M_i is the buffer of received messages. In the pseudocode, $M_i = d$ implies that the process p_i received the broadcast information d . The principle of 2RB is quite simple: The 2RB algorithm consists of 2 rounds. At the first round, the source process broadcasts the information d to all processes. At round 2, all processes that received d in previous rounds broadcast the information d .

The following lemma shows that the 2RB correctly broadcasts messages with high probability if no crash occurs during its execution.

Lemma 1 Let p_i be the process that broadcasts an information using 2RB, and n' be the number of non-crashed processes at the beginning of 2RB. If no process crashes during the execution of 2RB, all non-crashed processes receive the information d with probability $1 - O(n'/\alpha^{n'})$, where α is a constant more than one.

Proof Let X_1 and X_2 be the random variables that represent the number of processes receiving d by the end of rounds one and two, respectively. Since each process receives d with the probability more than $1/2$ at round one, we have $E[X_1] \geq n'/2$. Using the Chernoff Bound, we can obtain the following inequality;

$$\Pr \left[X_1 < n'/4 \right] \leq e^{-\frac{n'}{2} \cdot \frac{1}{8}} \leq e^{-\frac{n'}{16}}. \quad (1)$$

This implies that at the end of round one, at least $n/4$ processes receive d with high probability. Next, under the assumption of $X_1 \geq n'/4$, we bound the probability of $X_2 = n'$.

$$\begin{aligned} \Pr[X_2 = n' | X_1 \geq n'/4] &\geq \left(1 - \left(\frac{1}{2} \right)^{n'/4} \right)^{3n'/4} \\ &\geq 1 - \frac{3n'}{4} \left(\frac{1}{2} \right)^{n'/4} \\ &\geq 1 - \frac{3n'}{2^{n'/4+2}}. \end{aligned} \quad (2)$$

From the inequalities 1 and 2, we obtain $\Pr[X_2 = n'] \geq (1 - O(n'/\alpha^{n'}))$ for some $\alpha > 1$. \square

Let us consider the case where all n' non-crashed processes concurrently broadcast informations using 2RB. Let $d_i (0 \leq i \leq n' - 1)$ be the information that is broadcast by p_i . The following lemma gives the bound for the probability that all processes receive all informations $d_0, d_1, \dots, d_{n'-1}$.

Lemma 2 Let each process p_i ($0 \leq i \leq n' - 1$) broadcasts an information d_i using 2RB. If no process crashes during the execution of 2RB, all non-crashed processes receive all informations $d_0, d_1, \dots, d_{n'-1}$ with probability $1 - O(n'^2/\alpha^{n'})$, where α is a constant more than one.

Proof Let E_i be the event that all processes receives d_i , and \bar{E}_i be the complement of E_i . From Lemma 1, $\Pr[\bar{E}_i] \leq O(n'/\alpha^{n'})$ holds. Then, using the union bound, we obtain

$$\begin{aligned} \Pr \left[\bigcup_{i=1}^{n'-1} \bar{E}_i \right] &\leq \sum_{i=1}^{n'-1} \Pr[\bar{E}_i] \\ &\leq n' \cdot O(n'/\alpha^{n'}) \leq O(n'^2/\alpha^{n'}). \end{aligned} \quad (3)$$

Therefore, $\Pr[\bigcap_{i=1}^{n'-1} E_i] \geq 1 - O(n'^2/\alpha^{n'})$ holds. The lemma is proved. \square

4.2 Algorithm CPO

Algorithm Design Using the 2RB primitive, we present an algorithm CPO that solves the consensus problem under the $(1/2)$ -probabilistic omission adversary. Figure 2 is a pseudocode description of the algorithm. The algorithm CPO is designed using the *rotating coordinator* paradigm: An execution of CPO is divided into consecutive *cycles* that are identified by $0, 1, 2, \dots$. A coordinator is assigned to each cycle. In the k -th cycle of the execution, process $p_{(k \bmod n-1)}$ works as the coordinator. In the algorithm CPO, each process maintains an candidate of decision values. In Figure 2, the candidate value of p_i is stored in variable e_i . At initial configurations, the value of candidate variable e_i is p_i 's proposal v_i . Each candidate variable e_i has a timestamp t_i , which stores the cycle number when the last update of e_i occurs. In each cycle, the coordinator tries to make all processes reach agreement by imposing its own candidate value to all other processes. One cycle consists of two phases, and one phase consists of two rounds. The first phase of each cycle is called *impose phase*, and the second one is called *commit-aggregation phase*. In impose phases, only the coordinator broadcasts e_i using 2RB to tell its candidate value to all other processes. If a process p_i receives the message with the candidate value e from the coordinator, it stores e to its candidate variable e_i , and also updates its timestamp t_i with the current cycle number. In commit-aggregation phases, each process p_i broadcasts the information (e_i, t_i) using 2RB. Commit-aggregation phases have two role: The first role is that each process checks whether more than $n/2$ processes have the same candidate value as one owned by itself or not. If a process p_i passes the check, it decides its candidate value. The second role is that the coordinator of the next cycle aggregates candidate values. If the process $p_{(r \bmod n-1)}$ aggregates more than $n/2$ candidate values at the commit-aggregation phase of cycle $r-1$, it updates its candidate value with one having the latest timestamp of all received candidates. At the next cycle, $p_{(r \bmod n-1)}$ can actually work as the coordinator only if it receives more than $n/2$ candidate values. Otherwise, it will be silent at the next cycle. In Figure 2, the variable act_i is the flag whether the coordinator p_i works or not.

Correctness We prove the correctness of the algorithm CPO. We first introduce several notations used in the following proofs: Let $cod(c)$ be the identifier of the coordinator of cycle c (i.e., $cod(c) = (c \bmod n - 1)$). We define act_i^c , e_i^c and t_i^c as the values of act_i , e_i , and t_i at the beginning of a cycle c , respectively. We say that “the coordinator of c is active” if $p_{cod(c)}$ does not crash at the beginning of c and $act_{cod(c)}^c = \text{TRUE}$ holds. We say “a cycle c is valid” if no crash occurs during c and the coordinator of c is active. The cycle that is not valid is called *invalid*. For lack of space, several proofs are omitted (Lemma 4 and 6).

Lemma 3 Letting p_i be the process that decides at cycle c , the decision value of p_i is $e_{cod(c)}^c$.

Proof Let d be the decision of p_i . Since p_i decides at the cycle c , it receives the message (d, c) from the coordinator of cycle c . This implies $e_{cod(c)}^c$. \square

Algorithm CPO: Code for p_i

```

1: variable:
2:    $e_i$  : init  $v_i$  /* The  $v_i$  is the  $p_i$ 's proposal */
3:    $t_i$  : init  $-1$ 
4:    $act_i$  : init FALSE

5: for each cycle  $c = 0, 1, 2, \dots$  do :
6:   Impose Phase:
7:   if  $(c \bmod n - 1) = i$  and  $act_i = \text{TRUE}$  then 2RB( $e_i$ ) endif
8:   if a message is received from  $p_{c \bmod n - 1}$  then
9:     let  $e$  be the message from the current coordinator
10:     $e_i \leftarrow e$ ;  $t_i \leftarrow c$ 
11:   Commit-aggregation Phase:
12:   2RB( $e_i, t_i$ )
13:   if messages are received from more than  $n/2$  processes then
14:      $act_i \leftarrow \text{TRUE}$ 
15:     if  $(c \bmod n - 1) = i$  then
16:       let  $(e', t')$  be the message with the latest timestamp
17:        $e_i \leftarrow e'$ ;  $t_i \leftarrow c$ 
18:     endif
19:     if  $\exists w : (w, c)$  is received more than  $n/2$  times then
20:        $e_i \leftarrow w$ ;  $t_i \leftarrow c$ 
21:       decide( $e_i$ )
22:     endif
23:   else  $act_i \leftarrow \text{FALSE}$  endif
24: endfor

```

Fig. 2. Algorithm CPO: A consensus algorithm under $(1/2)$ -POA

Lemma 4 Let p_i be the process that decides a value d at cycle c . Then, for a process p_j and any cycle c' ($c' \geq c$), if $t_j^{c'} \geq c$, then $e_j^{c'} = d$.

Lemma 5 (Uniform Agreement) No process decides differently.

Proof Let p_i be the process that decides first (if two or more processes decide at a same round, arbitrary one of them is chosen), d be the decision value of p_i , and c be the cycle when p_i decides. Then, we prove that a process p_j ($\neq p_i$) also decides d . From Lemma 4, any cycle $c' (\geq c)$ whose coordinator is active, $e_{\text{cod}(c')}^{c'} = d$ holds. Since each process can decide at cycle x only if x 's coordinator is active, at the cycle y when p_j decides, y 's coordinator has the candidate value d at the beginning of y . From Lemma 3, this implies that p_j decides d . The lemma holds. \square

Lemma 6 (Validity) If a process decides a value d , then, d is a value proposed by a process.

Lemma 7 If cycles c and $c + 1$ are valid, all processes decide by the end of $c + 1$ with probability $1 - O(n^2/\alpha^n)$ (α is a constant).

Proof Clearly, all processes decide at the end of $c + 1$ if the following three events occur:

- The coordinator of $c + 1$ becomes active, i.e., in the commit-aggregation phase of c , the coordinator of $c + 1$ receives messages from more than $n/2$ processes.
- In the impose phase of $c + 1$, all non-crashed processes receive the message from the coordinator.
- In the commit-aggregation phase of $c + 1$, all non-crashed processes receive more than $n/2$ messages.

Since c and $c + 1$ are valid, from Lemma 2, each of the above three events occurs with a probability higher than $1 - O(n^2/\alpha^n)$. Thus, all processes decide at the end of $c + 1$ with probability $(1 - O(n^2/\alpha^n))^3 = 1 - O(n^2/\alpha^n)$. \square

Lemma 8 (Probabilistic Termination) Every correct process decides with probability 1. In addition, the worst-case expected number of rounds until all correct processes decide is $O(f)$.

Proof We call two consecutive valid cycles a *block*. Two blocks are said to be *independent* if there is no cycle belonging to both blocks. Since at most f processes can crash, for any $k > 0$, the execution consisting of $4f + 2k$ cycles includes k blocks independent with each other (notice that if a process p_j crashes during cycle c , it does not only makes the cycle c invalid, but also makes the cycle where p_j is coordinator invalid, i.e., one crash failure may make two cycles invalid). From Lemma 7, In every block, all non-crashed processes decide with probability $1 - O(n^2/\alpha^n)$. Then, the probability that there exists the process that does not decide at the end of the cycle $4f + 2k$ is $(O(n^2/\alpha^n))^k$, which converge to zero. Thus, every correct process decides with probability 1. In addition, The worst case expected number of cycles until all correct processes decide is $\sum_{k=1}^{\infty} (4f + 2k) \cdot (1 - O(n^2/\alpha^n)) \cdot O(n^2/\alpha^n)^{k-1} = O(f)$. Since one cycle consists of four rounds, we proved the $O(f)$ worst-case expected round complexity. \square

5 The $(f + O(1))$ -Round Algorithm for $t < n/3$

In this section, we show that a faster algorithm F-CPO, which achieves $f + O(1)$ time complexity under the assumption of $t < n/3$.

Algorithm Design We present an algorithm F-CPO in Figure 3. The algorithm F-CPO is based on the floodset algorithm, which is a traditional crash-tolerant consensus algorithm [12]. Same as CPO, the algorithm F-CPO maintains candidate variables at each process, and its execution is divided into consecutive cycles that are identified by $1, 2, \dots$. However, unlike CPO, one cycle of F-CPO consists

Algorithm F-CPO: Code for p_i

```

1: variable:
2:    $e_i$  : init  $v_i$  /* The  $v_i$  is the  $p_i$ 's proposal */

3: for each cycle  $c = 1, \dots$  do :
4:   2RB( $e_i$ )
5:   let  $E_i$  be the multiset of all received messages
6:   if  $|E_i| > 2n/3$  then
7:     if  $E_i$  includes only one value  $v$  then
8:       decide( $v$ )
9:     else
10:       $e_i \leftarrow$  the most frequent value in  $E_i$ 
11:    endif
12: endfor

```

Fig. 3. Algorithm F-CPO: A fast consensus algorithm on the system with $(1/2)$ -POA for $t < n/3$

of two rounds. In each cycle, all processes exchange their candidate values using 2RB primitive. At the end of each cycle, each process receiving sufficiently many number of messages updates its candidate variable by voting. More precicely, if a process p_i receives more than $2n/3$ candidate values, it updates its candidate variable with one that is received by p_i most frequently (if the frequencies of two or more values are same, the maximum one is choosen). In addition, if all received messages have a same value d , p_i decides d (but does not halt).

In the original floodset algorithm, one correct round (i.e., the round where no crash occurs) is sufficient to guarantee the termination because all non-crashed processes have a same candidate value after the correct round, and thus they decide at the following round. In contrast, since message omission is considered in our model, no-crash-failure round does not guarantee that all candidate values become a same one. However, in our model, one correct round (to be exact, one cycle including one correct round) triggers the agreement of all candidate values. Actually, in the algorithm F-CPO, it is guaranteed with high probability that all non-crashed processes have a same candidate if a cycle c includes one correct round and the number of crashes occuring the following cycle $c + 1$ is at most two. Notice that such pair of cycles necesarily exists in the execution of $f/2 + 1$ cycles, which is the key fact that our algorithm achives $f + O(1)$ expected round complexity (the detail is shown in the correctness proof).

Correctness We prove the correctness of the algorithm F-CPO. Let $E(c)$ be the multiset consisting of candidate values of all non-crashed processes at the beginning of cycle c , and $\#_x(c)$ be the times that value x appears in $E(c)$. We define $\text{1st}(c)$ as the value d that maximize $\#_d(c)$ (if there are two or more values that maximize $\#_d(c)$, the largest candidate value is chosen as $\text{1st}(c)$). We also

define $\text{2nd}(c)$ as the value d that secondly maximizes $\#_d(c)$ after $\text{1st}(c)$ (Same as $\text{1st}(c)$, if two or more values secondly maximize, the largest one is chosen). A number of proofs are omitted for lack of space (Lemma 9, 10, and 11).

Lemma 9 Assume that p_i decides a value d at cycle c . If a process p_j ($\neq p_i$) updates its candidate variable e_j at cycle $c'(\geq c)$, then its updated value is d .

Lemma 10 No process decides differently.

Proof Using the lemma 9, we can prove this lemma by the same way as the lemma 5. \square

Lemma 11 If a process decides a value d , then, d is a value proposed by a process.

Lemma 12 Let n' be the number of non-crashed processes at the beginning of c , and p_j be the process that broadcast an information d using 2RB. Then,

1. if p_j does not crash at round $2c - 1$ (that is, the first round of the cycle c), and at most one process crashes at round $2c$ (the second round of c), the information d is received by all non-crashed processes with probability $1 - O(n/\alpha^n)$, and
2. if p_j crashes at the first round of c and no process crashes at the second round of c , the information d is received by more than $n'/2 + 2$ or less than $n'/2 - 2$ processes with probability $1 - O(n^{-1/2})$.

Proof (Proof of 1) This lemma can be proved in the same way as Lemma 1. **(Proof of 2)** Let x be the number of processes that receive d at the beginning of round $2c$ (notice that the value x is not the random variable because the set of messages sent by p_j at round $2c - 1$ is deterministically controlled by the adversary). Clearly, if $x = 0$, no process receives messages d at the end of c . Thus, in the followings, we consider the case of $x > 0$. We define P_1 as the set of processes that receive d with probability 1 (i.e., any process in P_1 receives the message with d that is not contained in the omission candidate C_{2c}). We also define n_1 as the cardinality of P_1 . If $n_1 > n'/2 + 2$ holds, $n'/2 + 2$ processes necessarily receive d and thus the lemma holds. Thus, we assume $n_1 \leq n'/2 + 2$. Since there is no crash at round $2c$, any non-crashed process not in P_1 unreceives the information d with probability $(1/2)^x$ (denoted by q for short). Letting X as be the random variable representing the number of processes receiving d at the end of cycle c , we obtain

$$\Pr[X = k] = \binom{n' - n_1}{n' - k} q^{n' - k} (1 - q)^{k - n_1}. \quad (4)$$

Since this is the binomial distribution, it takes the maximum in the average case of $n' - k = (n' - n_1)q$. Thus,

$$\begin{aligned}\Pr[X = k] &\leq \binom{n' - n_1}{(n' - n_1)q} q^{(n' - n_1)q} (1 - q)^{(n' - n_1)q} \\ &\leq O(n^{-1/2}),\end{aligned}\tag{5}$$

where we use a weaker variant of Stirling approximation $\binom{N}{qN} < q^{-qN} (1 - q)^{-N(1-q)} \cdot O(N^{-1/2})$ and the fact of $(n' - n_1) = \Theta(n)$. From this inequality, using the union bound, we have $\Pr[(n'/2 - 2) \leq X \leq (n'/2 + 2)] \leq \sum_{z=-2}^2 \Pr[X = (n'/2 + z)] \leq 5 \cdot O(n^{-1/2})$. The lemma is proved. \square

Lemma 13 Let c be a cycle where at most one process crashes. Then, $|\#_{1st}(c + 1) - \#_{2nd}(c + 1)| > 2$ holds with probability $1 - O(n^{-1/2})$.

Proof If no crash occurs, the lemma clearly holds from Lemma 2 because all non-crashed processes receives a same set of messages with probability $1 - O(n^2/\alpha^n)$. Thus, we only consider the case where one process crashes. Let $\hat{p}_0, \hat{p}_1, \dots, \hat{p}_{n'-1}$ be non-crashed processes at the end of c , and E_i be the event that all n' processes receives the information sent by \hat{p}_i . The process that crashes during c is denoted by p_x . We also introduce the event E_X that the information broadcast by p_x is received more than $n'/2 + 2$ or less than $n'/2 - 2$ processes. For a event E , we denote its complement by \bar{E} . From Lemma 12, $\Pr[\bar{E}_i] \leq O(n/\alpha^n)$ and $\Pr[\bar{E}_X] \leq O(n^{-1/2})$ holds. Then, using the union bound, we obtain

$$\begin{aligned}\Pr\left[\left(\bigcup_{k=0}^{n'-1} \bar{E}_i\right) \cup \bar{E}_X\right] &\leq \sum_{k=0}^{n'-1} \Pr[\bar{E}_i] + \Pr[\bar{E}_X] \\ &\leq (n' - 1)O(n/\alpha^n) + O(n^{-1/2})\end{aligned}\tag{6}$$

Therefore, $\Pr[(\bigcap_{k=1}^{n'-1} E_i) \cap E_X] \geq 1 - O(n^{-1/2})$ holds. This implies that at least $n'/2 + 2$ processes receive a same set of messages consisting of more than $2n/3$ informations (notice $n' > 2n/3$) with probability $1 - O(n^{-1/2})$. Then, more than $n/2 + 2$ processes update their candidate variables with a same value. It follows that $|\#_{1st}(c + 1) - \#_{2nd}(c + 1)| > 2$ holds with probability $1 - O(n^{-1/2})$. The lemma is proved. \square

Lemma 14 Let c be a cycle where at most two processes crash, and n' be the number of non-crashed processes at the beginning of c . Then, with probability $1 - O(n^2/\alpha^n)$, all processes that do not crash at the end of c receive at least $\max\{n' - 2, 2n/3\}$ messages.

Proof Using Lemma 12, we can prove this lemma in the same way as Lemma 2. \square

Lemma 15 Assume $|\#_{1st}(c) - \#_{2nd}(c)| > 2$. If at most two processes crash during cycle c , all non-crashed processes have a same candidate value with probability $1 - O(n^2/\alpha^n)$.

Proof Let n' be the number of processes that do not crash at the beginning of c . From Lemma 14, we can guarantee, with probability $1 - O(n^2/\alpha^n)$, that all non-crashed processes receive $\max\{n'-2, 2n/3\}$ messages. Since $|\#_{1st}(c) - \#_{2nd}(c)| > 2$ holds, the most frequent value of all received by each process is $1st(c)$. This implies that all processes update its candidate variable with the value $1st(c)$. The lemma is proved. \square

Lemma 16 (Probabilistic Termination) Every correct process decides with probability 1. In addition, the worst-case expected number of rounds until all correct processes decide is $f + O(1)$.

Proof We define a *block* as two consecutive cycles where the first cycle have at most one crash, and the second one has at most two crashes, respectively. Two blocks are said to be independent if there is no cycle belonging to both blocks. Let X_1 be the random variable representing the number of cycles until all processes have a same candidate value, and X_2 be the random variable representing the number of cycles taken by the decision of all processes from the cycle when all processes have a same candidate value. We first bound $E[X_1]$ under f_1 process crashes. For any $k > 0$, the execution consisting of $f_1/2 + 1 + 2k$ cycles includes $k + 1$ blocks independent with each other. From Lemmas 13 and 15, in every block, all non-crashed processes decides with probability $1 - O(1/\sqrt{n})$. Thus, we obtain $E[X_1] = \sum_{k=1}^{\infty} (f_1/2 + 1 + 2k) \cdot (1 - O(1/\sqrt{n})) \cdot O(1/\sqrt{n})^k = f_1/2 + O(1)$. Next, assuming f_2 crash failures are possible, we bound $E[X_2]$. From Lemma 14, if a cycle c has only two crash processes, all processes decides at the end of c with probability $1 - O(n^2/\alpha^n)$. Thus, we obtain $E[X_2] = \sum_{k=1}^{\infty} (f_2/2 + 1 + 3k) \cdot (1 - O(n^2/\alpha^n)) \cdot O(n^2/\alpha^n)^k = f_2/2 + O(1)$. These two bounds implies that the expected number of cycles until all correct processes decide is $E[X_1] + E[X_2] = (f_1 + f_2)/2 + O(1) = f/2 + O(1)$. Since one cycle consists of two rounds, we have proved the $f + O(1)$ worst-case expected round complexity. \square

6 Concluding Remarks

This paper has newly introduced a novel class of adversaries, called p -probabilistic omission adversary (p -POA). We also investigated the consensus algorithm working under p -POA, and its complexity. We proposed two impossibility results. The first one is that no algorithm can tolerate more than of equal to $n/2$ crash failures, for any $p > 0$. The second one is that for any $p > 0$, no algorithm can halt. We proposed two consensus algorithms CPO and F-CPO working under the $(1/2)$ -probabilistic omission adversary. These algorithms have distinct advantages in the point of crash-failure resiliency and time complexity respectively.

References

1. H. Attiya and J. L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.
2. A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *Proc. of 10th International Workshop on Distributed Algorithms(WDAG)*, volume 1151 of *LNCS*, pages 105–122, 1996.
3. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
4. C. Delporte-Gallet, H. Fauconnier, and F. C. Freiling. Revisiting failure detection and consensus in omission failure environments. In *Proc. of Second International Colloquium on Theoretical Aspects of Computing(ICTAC)*, volume 3722 of *LNCS*, pages 394–408, 2005.
5. D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.
6. D. Dolev, R. Reischuk, and R. Strong. Early stopping in byzantine agreement. *Journal of ACM*, 37(4):720–741, 1990.
7. 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, 1985.
8. F. C. Freiling, M. Herlihy, and L. D. Penso. Optimal randomized fair exchange with secret shared coins. In *Proc. of the 9th International Conference on Principle of Distributed Systems(OPODIS)*, volume 3974 of *LNCS*, pages 61–72, 2005.
9. R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and consensus. In *Proc. of 9th International Workshop on Distributed Algorithms(WDAG)*, volume 972 of *LNCS*, Sep 1995.
10. V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.
11. M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13:124–149, 1991.
12. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
13. Y. Moses and S. Rajsbaum. A layered analysis of consensus. *SIAM Journal on Computing*, 31(4):989–1021, 2002.
14. G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, 1990.
15. P. R. Parvédy and M. Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In *Proc. of the 16th annual ACM symposium on Parallelism in algorithms and architectures(SPAA)*, pages 302–310, Jun 2004.
16. K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, SE-12(3):477–482, 1986.
17. M. Raynal. Consensus in synchronous systems: A concise guided tour. In *Proc. of Pacific Rim International Symposium on Dependable Computing(PRDC)*, pages 221–228, 2002.
18. N. Santoro and P. Widmayer. Majority and unanimity in synchronous networks with ubiquitous dynamic faults. In *Proc. of the 12th International Colloquium on Structural Information and Communication Complexity(SIROCCO)*, volume 3499 of *LNCS*, pages 262–276, May 2005.