A Generalized Byzantine Model

Wang Cheng [*] Carole Delporte-Gallet [†] Hugues Fauconnier [‡]
Rachid Guerraoui [§] Anne-Marie Kermarrec [¶]

May 2, 2014

Abstract

The classical Byzantine model assumes that a process is either correct and obeys the protocol assigned to it, or is Byzantine and may behave completely arbitrarily, sending corrupted messages to all processes during its entire execution. In this paper we generalize that model and consider that, in each communication step, some of the processes might send corrupted messages to a *subset* of the processes. This enables to capture more accurately practical situations where processes experience possibly temporary bugs in specific parts of their code. We present this model and prove bounds on the number of correct processes needed to solve the classical interactive consistency and consensus problems.

This paper is a regular submission.
The paper is a student paper.

[*]École Polytechnique Fédérale de Lausanne,Switzerland, cheng.wang@epfl.ch, +41 78 924 08 90.
[†]LIAFA-Université Paris-Diderot, Paris, France, cd@liafa.univ-paris-diderot.fr, +33 1 57 27 92 25.
[‡]LIAFA-Université Paris-Diderot, Paris, France, hf@liafa.univ-paris-diderot.fr, +33 1 57 27 92 25.
[§]École Polytechnique Fédérale de Lausanne, Switzerland, rachid.guerraoui@epfl.ch, +41 21 693 52 72.
[¶]INRIA Rennes Bretagne-Atlantique, France, anne-marie.kermarrec@inria.fr, +33 2 99 84 25 98.

1 Introduction

Pease, Shostak and Lamport were the first to introduce the Byzantine model in their landmark paper [10, 13]. In their model, a Byzantine process is defined as a process that can arbitrarily deviate from the protocol assigned to it. They proved that agreement is achievable with a fully connected network if and only if the number of Byzantine processes is less than one third of the total number of processes. Dolev extended this result to general networks, in which the connectivity number is more than twice the number of faulty processes [2]. The early work on Byzantine agreement is well summarized in the survey by Fisher [5].

Many approaches have been proposed to circumvent the impossibility results of [7] in the Byzantine context. The work presented in [3] introduced the concept of partial synchrony: an intermediate model between synchronous and asynchronous models, allowing some limited periods of asynchrony. Partial synchrony is considered weak enough to model real systems while strong enough to make Byzantine agreement solvable. Alternative approaches rely on randomized algorithms (e.g. [1, 4, 9, 15]).

Accounting for the fact that communication failures sometimes dominate computation ones (due to high reliability of hardware and operating systems), other models focus on communication failures (e.g. [14, 18]) and hybrid failures (e.g. [8, 11]). Such models consider that the Byzantine components are the communication channels instead of (in addition to) the processes. For instance, in [16, 17], Santoro and Widmayer showed that consensus cannot be achieved with $\lceil \frac{n}{2} \rceil$ Byzantine communication faults.

In the classical Byzantine failure model, as well as in models with Byzantine communication channels, the notion of Byzantine failure is sustainable over time: the models consider that the processes (resp. the channels) are either Byzantine, or are correct for the entire duration of the computation. More specifically, the Byzantine failure model encompasses two different situations: (1) A system attack where an adversary coordinates the behavior of several processes (resp. channels) in order to corrupt the system (e.g. denial of service attack); (2) Software or hardware bugs that lead one or several processes (resp. channels) to behave in an arbitrary manner. We argue that assumptions are fairly different in these two situations. Typically, under attack, the set of Byzantine processes might indeed send corrupted messages systematically, possibly with the sole purpose of corrupting the entire execution of the algorithm. In situations where processes misbehave due to bugs, the number of corrupted messages might vary over time. In this case, the Byzantine failure model as defined in [10, 13] can be viewed as too conservative. It assumes the worst: failures occur due to a malicious intent rather than simply arbitrary bugs located in specific parts of the code that might not be repeated.

In this paper, we generalize the Byzantine failure model, accounting for these differences and reconsider some of the theoretical impossibilities in this light. We study our generalized Byzantine model in a synchronous context of n processes of which up to m can be faulty. The processes can communicate with each other directly through a complete network. We assume that each faulty process (partially controlled by the adversary) is associated with up to t Byzantine communication links. We call such a process the *t-faulty process*. These t Byzantine links are *dynamic*: they may be different in different communication rounds. Note that if $t = n - 1$, our generalized Byzantine model instantiates the classical Byzantine model. From the component failure model's view, our generalization is orthogonal to those of [21] or [17]. When $m = t$, our model is a pure communication failure model.

In this paper, we focus on the case when $t < n$. The most important feature in this case is that *t-faulty* processes always send correct messages to some processes in the system, which enables us for instance to solve the *interactive consistency* problem [5]. This problem consists in devising an algorithm that allows every process p to decide a value for each process q, such that: (1) If p and q are correct, then p decides the initial value of q. (2) All correct processes decide the same value for each process. Somehow, $t < n$ implies that the local computation of the processes is always correct and only the communication links related to the faulty processes are partially controlled by the adversary - during specific rounds. In this sense, we treat all the processes as correct ones. In this context *interactive consistency* means that every process knows the initial value of every other process by the end of the computation.

The results obtained are summarized in Table 1. We give the necessary and sufficient conditions for reaching interactive consistency with oral and signed messages. Our sufficiency proofs are constructive and the algorithms we provide are deterministic, while the necessity proofs are based on scenario argument.

Oral messages	$n > \max\{2m + t, 2t + m\}$
Signed messages	$n > 2t + m$

Table 1: Necessary and sufficient conditions for solving interactive consistency with t-faulty failure.

The rest of the paper is organised as follows. Section 2 describes our system model and the agreement problems to be researched. In section 3 (resp. section 4), the tight bound for solving interactive consistency problem with respect to oral messages (resp. signed messages) is discussed. We then investigate the consensus problem in section 5, before concludes the paper in seciton 6.

2 Model and Definitions

We consider a synchronous message-passing distributed system P of n processes. Each process is identified by a unique id $p \in \{1, \ldots, n\}$. As in [10, 20], a *synchronous computation* proceeds in a sequence of *rounds*. The nodes communicate with each other by sending messages round by round within a fully connected point-to-point network. In each round, every process first sends at most one message to every other process, possibly to all processes, and then p receives the messages sent by other processes. The communication channels are authenticated, i.e., the sender is known to the recipient.

Each process has an input register with its initial value from some domain D, and an output register which records the outcome of the computation. Note that the output register can be written at most once. When a process writes its output register, we say that the process decides. We model an algorithm as a set of deterministic automata, one for every process in the system. Thus, the actions of a process are entirely determined by the algorithm, the initial value and the messages it receives from others. In this paper we assume that processes always follow their protocol.

2.1 Failure model

In short, a faulty process p may lie to other processes: even if p follows its code, p can send to a subset of processes Byzantine messages, i.e., messages that result from p not following the protocol.

More precisely, we assume an adaptive (or dynamic) adversary which introduces Byzantine faults in transmission: these faults may only come from faulty processes. Here, up to m of the processes are faulty and controlled by the adversary. In each round, the adversary chooses up to t communication links from each faulty process that will carry Byzantine messages.

An instance of our model of n processes with m faulty processes such that in each round up to t communication faults on links from faulty processes may occur will be called an (n, m, t)*-system*. For the results established in this paper, we always assume $m > 0$, $t > 0$, and $n > max\{m, t\}$. But the model itself does not preclude other cases. We also assume $n > 1$, the case $n = 1$ being trivial.

We consider two authentication cases: *oral messages* and *signed messages*. Following [10, 19], with oral messages, the sender is always authenticated by the receiver. But contrary to messages with unforgeable signatures, a process q may make process p believe that q has received message m from process r even if it is not true. For a signed message, the sender attaches its signature to the messages. A signed message satisfies the two following properties:

a) A correct process's signature cannot be forged and any alteration of the content of its signed messages can be detected.

b) Any process can verify the authenticity of a process's signature.

Note that the signature of a faulty process can be forged by another faulty process.

2.2 Full information protocols

To prove our impossibility results, we consider full information protocol as in [6, 10, 12]. In a full information protocol, every process transmits to all processes in each round everything it knows about all the values sent by other processes in the previous round.

We use $P^{l:k}$ to denote the set of strings of symbols in P of length at least l and at most k, P^+ to denote nonempty strings of symbols in P and P^* to denote all the strings including the empty one.

A *k-round scenario* (for (n, m, t)-system P) describes an execution of the protocol. Intuitively σ describes a communication scheme admissible for a (n, k, m) system. It gives the initial value of each process and the communication scheme. It captures the outcome of a k-round full information exchange. Given scenario σ, $\sigma(p_1 p_2 \dots p_k)$ is intended to represent the value p_{k-1} tells p_k that p_{k-2} tells p_{k-1} ... that p_1 tells p_2 is p_1's initial value.

More specifically, a *k-round scenario* σ is a mapping :$P^{1:k+1} \to D$, such that:

- For a string p of length 1, $\sigma(p)$ is p's initial value.
- There is a partition of P into two sets R_σ, the set of correct processes, and U_σ, the set of faulty ones such that:
  - $|U_\sigma| \leq m$ (and then $|R_\sigma| \geq n - m$)
  - for every process $p \in R_\sigma$: $\sigma(wpq) = \sigma(wp)$ for all $q \in P$ and $w \in P^{0:k-1}$,
  - for every process $p \in U_\sigma$, for every round j in $\{1, \dots, k\}$, there is a set T of at most t processes such that for all $q \in P \setminus T$ and for all $w \in P^{0:k-1}$ we have $\sigma(wpq) = \sigma(wp)$.

Note that if $\sigma(wpq) \neq \sigma(wp)$ for some strings w of length l, that means that q receives a Byzantine message from p in round l.

If σ is a *k-round scenario* and $p \in P$, p's view of σ is the map σ_p defined by $\sigma_p(w) = \sigma(wp)$.

Let O be the set of possible outputs and $\mathcal{U}^k$ be the set of mappings from P^k into D. Any k-round algorithm $\mathcal{A}$ defined in an (n, m, t)-system may be defined on the set of all scenarios; namely as a set $\{F_p : p \in P\}$ of functions, where $F_p : \mathcal{U}^k \to O$. Then without loss of generality, we consider in the following only algorithms defined this way on the set of all scenarios by a set $\{F_p : p \in P\}$ of functions.

$\mathcal{A}$ solves a problem if for each k-round scenario σ and every process $p \in P$, $F_p(\sigma_p)$ satisfies the specification of the problem.

2.3 Problem specifications

We will first address the problem of *interactive consistency* [5]. In this case, each process maintains an output register of n entries. The j-th entry of the output register of a process i contains the value decided by process i for process j. It is defined by the two following properties:

- *Termination*: Eventually, every process decides a value for each process.
- *Validity*: If process p decides v for process q, then v should be the initial value of q.

As a remark, validity implies that all the processes decide the same values. Note that, contrary to classical models with faulty processes, here even faulty processes have to decide.

Let $\mathcal{A} = \{F_p : p \in P\}$ be a k-round algorithm and the output is a vector that contains the n decided values then O is $(D \cup \{\perp\})^n$. Then $\mathcal{A}$ solves interactive consistency if for each k-round scenario σ and every process $p \in P$, $F_p(\sigma_p)$ satisfies the Termination and Validity properties of interactive consistency, i.e. for each k-round scenario σ and all $p, r \in P$ we have $F_p(\sigma_p)[r] = \sigma(r)$.

We will also consider the *binary consensus* problem [5]. In this problem, we assume that every process's input value is in $\{0, 1\}$ and its output register contains $\perp$, 0 or 1. $\perp$ means that the register has not yet written by the process. The value written in the output register is named the decision value. The *binary consensus* problem is defined by the three following properties:

- *Termination*: Eventually, every process decides a value.
- *Agreement*: Every two processes decide the same value.
- *Validity*: If all processes start with the same initial input, then every process should decide this value.

Let $\mathcal{A} = \{F_p : p \in P\}$ be a k-round algorithm and the output is in $\{0, 1, \perp\}$. Then $\mathcal{A}$ solves binary consensus if for each k-round scenario σ and every process $p \in P$, $F_p(\sigma_p)$ satisfies the Termination, Agreement and Validity properties of binary consensus, i.e. if for each k-round scenario σ, there exists v the initial value of some process such that for all processes p and r, we have $F_p(\sigma_p) = v$, where v is the initial value of some process.

Algorithm 1 OMIC

OMIC(0):

1. Every process broadcasts its initial value.
2. Every process receives the messages of the round. Let v_j^i be the value received by i from j, v_i^i is the initial value of i.
3. Process i decides v_j^i as the initial value of process j.

OMIC(k), $k > 0$:

1. Each process acts as a transmitter and sends its value to other $n-1$ processes (receivers).
2. Every receiver process uses the value it gets in step 1 as initial value, and executes OMIC($k-1$).
3. After running algorithm OMIC($k-1$) in step 2, every receiver process get a vector of values representing what other receivers get from the transmitter, and then uses the majority values as its decision on the initial value of the transmitter process.

3 Interactive Consistency With Oral Messages

In this section, we establish the tight bound for an (n, m, t)-system to solve *interactive consistency* with *oral messages*.

Theorem 1. *Interactive consistency can be achieved in an (n, m, t)-system with oral messages if and only if $n > \max\{2m + t, 2t + m\}$.*

The proof of the theorem contains two parts with a series of lemmas. First, we devise an algorithm we call OMIC to show the sufficiency of the theorem. And then we show the necessary part based on scenario argument.

3.1 Algorithm

Just like in [10], we use *transmitter* to denote the process that wants to transmit its value, and use *receiver* to denote the process that wants to know the value of the transmitter.

We first consider the correctness of a special case of the algorithm, and then the general case by induction. More precisely, we show that in an (n, m, t)-system, two rounds are enough to achieve interactive consistency as long as $n \geqslant 2(m + t)$. Additional rounds are necessary only when the number of processes is between $2(m + t) - 1$ and $\max\{2m + t, 2t + m\}$. Beyond that, Interactive consistency cannot be achieved.

Lemma 1. *If $n \geqslant 2(m+t)$, then the protocol OMIC(1) achieves interactive consistency in 2 rounds.*

Proof. Let us fix a process p as a transmitter. If we prove that each process decides the initial value of p, then the lemma is proved.

First suppose p is correct, then all the receivers in the first round will get the initial value of p. In the second round when applying OMIC(0), every receiver gets at most m wrong values for p

due to the m faulty processes. As $n-1>2m$, each receiver decides the correct initial value of the transmitter.

If p is faulty, up to t receivers will get wrong messages in the first round. Then in the second round, running OMIC(0), every receiver gets at most $t+m-1$ wrong initial values for p, $m-1$ from the other faulty processes and t from the receivers that have received wrong messages. Since $n \geqslant 2(t+m)$, $n-1>2(t+m-1)$, the receiver receives a majority of the correct value and decides correctly. □

We now consider the case where additional rounds are necessary to achieve interactive consistency.

Lemma 2. *For any $k \geqslant 1$, if $n>2m+k$, if the transmitter is correct then every receiver in OMIC(k) decides on the initial value of the transmitter.*

Proof. The proof is by induction on k. Suppose the transmitter is correct. The lemma is trivial for $k=1$. In the first round, all receivers get the initial value of the transmitter. In the second round (i.e. applying OMIC(0)), every receiver gets at most m wrong values for p due to the m faulty processes. As $n-1>2m$, each receiver decides the correct initial value of the transmitter.

We assume now the lemma is true for $k-1$ ($k>0$), and prove it for k.

In the first step of the algorithm, the correct transmitter sends its value to the other $n-1$ receivers among which up to m are faulty. These receivers act as transmitters in OMIC($k-1$). By induction hypothesis, if a transmitter is correct, every receiver in OMOC($k-1$) decides on the initial value of the transmitter. Then each receiver in OMIC(k) gets at least $n-1-m$ copies of the initial value in Step 3. Since there are only m faulty processes, $n-1-m>m+k-1 \geqslant m$, a majority of the values obtained in Step 3 are the initial values. That is to say every process obtains the initial value of every correct process. □

Lemma 3. *For any $k \geqslant 1$, OMIC(k) ensures interactive consistency if $k \leqslant m$, $n>\min(2m+k, 2m+2t-k)$.*

Proof. The proof is also by induction on k. If $k=1$, the condition is equal to $n>2m+2t-1$. Therefore by Lemma 1, the present lemma is proved. Now suppose the lemma is correct for $k-1$. We prove it for k.

We first assume the transmitter is correct. Since $n>2m+k$, by lemma 2 OMIC(k) guarantees each process decides on the initial value of the transmitter. So we have to verify the case in which the transmitter is faulty.

If the transmitter is faulty, then at most $m-1$ of the receivers are faulty. Since $k-1 \leqslant m-1$, $n-1>2m+(k-1)$, and $n-1>2(m-1)+2t-(k-1)$, OMIC($k-1$) ensures interactive consistency by induction hypothesis. Every receiver knows the values that other receivers received in Step 1. As $n-1>2m+2t-k-1 \geqslant 2t$, the majority of the values the transmitter sent is the initial value of the transmitter. So all the processes decide the same correct value. □

Lemma 4. *If $m \geqslant t$ and $n>2m+t$, it is possible to achieve interactive consistency in an (n,m,t)-system in $t+1$ rounds.*

Proof. Since $m \geqslant t$ and $n>2m+t$, we have $n>2t+t$ and $n>2m+2t-t$. Take $k=t$, by Lemma 3 above, OMIC(t) ensures interactive consistency. □

Lemma 5. *If $t \geqslant m$ and $n > 2t + m$, it is possible to achieve interactive consistency in an (n, m, t)-system in $m + 1$ rounds.*

Proof. Since $t \geqslant m$ and $n > 2t + m$, we have $n > 2t + m$ and $n > 2m + 2t - m$. Take $k = m$, by lemma 3 above, OMIC(m) ensures interactive consistency. □

From the above two lemmas, it is easy to deduce the sufficient part of Theorem 1. If $m \geqslant t$ we get the result by Lemma 4 and if $m < t$ by Lemma 5.

3.2 Impossibility

Now let us turn our attention to the necessary property of the theorem. We study two cases: $m \geqslant t$ and $t \geqslant m$. For each case, we proceed by contradiction. We construct two scenarios such that there is a set of processes for which these two scenarios are indistinguishable. In the first scenario they have to decide 0 for that set of processes, while in the second scenario they have to decide 1.

Lemma 6. *If $m \geqslant t$ and $n \leqslant 2m + t$, it is impossible to achieve interactive consistency in an (n, m, t)-system with oral messages.*

Due to space limitations, the proof is moved to Appendix A.1. Here we describe the intuition behind the proof. P can be partitioned into three non-empty sets A, B, and C, with $|A| \leqslant m$, $|B| \leqslant m$, $|C| \leqslant t$.

We define two scenarios α and β. In α, all processes have 0 as initial values. Processes in B are faulty and send to processes C messages pretending that processes in A have 1 as initial value.

In β, processes in A have 1 as initial value and all others processes have 0 as initial value. Processes in A are faulty and send to processes in C messages pretending that processes in A have 0 as initial value (see Figure A.1).

In the two scenarios, processes in C get the same messages, but in the first scenario they have to decide 0 for processes in A and 1 in the second scenario leading to a contradiction.

Lemma 7. *If $t \geqslant m$ and $n \leqslant 2t + m$, it is impossible to achieve interactive consistency in an (n, m, t)-system with oral messages.*

The proof of this Lemma is similar to the above one. Due to space limitations, it is moved to Appendix A.2.

From the above two lemmas, it is easy to deduce the necessary property of Theorem 1. If $m \geqslant t$ we get the result by Lemma 6 and if $m < t$ by Lemma 7.

4 Interactive Consistency With Signed Messages

In this section we present the tight bound for an (n, m, t)-system to reach *interactive consistency* with *signed messages*. In this new setting, we have the following main result:

Theorem 2. *It is possible to achieve interactive consistency in an (n, m, t)-system with signed messages if and only if $n > 2m + t$.*

As in the previous section, we rely on 2 steps to prove this result. We provide an algorithm, resp. a counterexample, to prove the sufficient, resp. the necessary property.

The algorithm, called SMIC, is simple and terminates in 3 rounds.

Algorithm 2 SMIC

1. Every process broadcasts its initial values to all other processes (with its signature).
2. Every process broadcasts the messages received in step 1 with its signature to all processes.
3. Every process broadcasts the messages received in step 2 with its signature to all processes.

After this, every process i has received messages like $\sigma(jkli)$. For every j, k, if $\{\sigma(jkli)\}_l$ have the same value v for different l, then i appends v into V_j as the value k received from j. i decides for j the majority value in V_j.

4.1 Algorithm

In the algorithm, called SMIC, each process i maintains a list V_j, containing the possible initial values of process j.

Lemma 8. *If $n > m + 2t$, an (n, m, t)-system can reach interactive consistency with signed messages.*

Proof. If j is correct, all the values $\sigma(jkli)$ will be the initial value of j since the signature of j can not be forged. So all the entries in V_j are set to $\sigma(j)$, and the majority of V_j is still v.

If j is faulty, suppose the initial value is v, as $n > m + 2t$ at least $t + 1$ correct processes $\{k_1, ..., k_{t+1}\}$ will receive $\sigma(jk_s) = \sigma(j)$ equal to v. Since k_s is correct, $\{\sigma(jk_sli)\}_l$ are all v. This will contribute to at least $t+1$ values v in V_j. On the other hand, only when $\sigma(jk)$ is different from v, k can contribute different values in V_j, because every $\sigma(jk)$ is always correctly sent to a subset of correct processes. Since j can send at most t wrong values in the first broadcast, V_j contains at most t different values from v, which leads the majority value of V_j to be v. $\square$

Therefore the sufficient part of Theorem 2 is proved. Now let us move to the necessary part.

4.2 Impossibility

Note that the two scenarios that we used in the proof of Lemma 6 are now impossible. The processes in set B cannot send to processes in set C a message where they pretend that the initial values of processes in set A are not the initial value that processes in A have sent.

Lemma 9. *If $n \leqslant m + 2t$, it is impossible to achieve interactive consistency in an (n, m, t)-system with signed messages.*

The proof of this necessary property of Theorem 2 is similar to that of Lemma 6. Duo to space limitation, it is moved to Appendix A.3.

5 An Upper Bound For Consensus

In this section, we establish an upper bound for binary consensus with oral messages. In binary consensus, processes do not need to agree on the initial value of each process. It is enough that they agree on some value to reach agreement. So in our setting, binary consensus is easier than interactive consistency.

Algorithm 3 OMC

OMC(0):

1. Every process broadcasts its initial value.

2. Every process, for example i uses the value it receives from process $j(\neq i)$ as v_j, or uses $\varnothing$ if it receives nothing. Let v_i denote the initial value of i.

3. Take $Major([v_1, \ldots, v_n])$ as output of i.

OMC(k), $k > 0$:

1. Each process acts as a transmitter and sends its value to other $n-1$ processes (receivers).

2. Every receiver process uses the value it gets in step 1 as initial value, and executes OMC($k-1$) for one time to decide a value as a guessed initial value for the transmitter. After this, receiver i appends the guessed values into V_i. Note that there are $n-1$ transmitters different from i, so V_i now has $n-1$ values.

3. *Feedback step*: every receiver i sends its guessed initial value for transmitter j to j. j add the majority of the received values in this round into V_i as the replacement of its initial value.

4. Process i takes $Major(V_i)$ as its output.

Theorem 3. *It is possible to achieve binary consensus in an (n, m, t)-system with oral messages if $n > 2m + t$.*

We give a new algorithm OMC which is slightly different from OMIC. OMC achieves binary consensus. The basic idea in OMC consists in adding a *feedback step* to inform the transmitter about the guessed value other processes agreed on. The agreement is also based on majority selection. When n is only greater than $2m + t$ (not $\max\{2m + t, 2t + m\}$), the majority value in algorithm OMIC may be several different values. So we assume a selection function *Major* to fix such situations. *Major* takes a list as input. The requirement we make for this function is that it alway returns the majority value in the input list. If there are several values with the same highest count in the input, *Major* will return one of them. The function is deterministic. The same input list (the order of the elements can be different) leads to the same output. Each process i maintains a list V_i to record the guessed values for all processes.

As we already showed in Section 3, we cannot guarantee interactive consistency with OMIC with only $n > 2m+t$. However we ensure interesting property in OMC with respect to $n > 2m+t$. More specifically, when $n > 2m + t$, we can prove that the elements in list V_i are the same for different i, but the values in V_i are not always equal to the initial values of the transmitters. We show this by induction as in Section 3.

Lemma 10. *If $n \geqslant 2m + 2t$, then OMC(1) achieves binary consensus in 2 rounds in an (n, m, t)-system.*

Proof. As we have shown in Lemma 1, the guessed value in Step 2 is the same as the initial value of transmitter when $n \geqslant 2m + 2t$. As the output of each process is the result of applying *Major* to the list consisting of the initial values the properties of binary consensus are satisfied. □

Lemma 11. *For any $k \geqslant 1$, if $n > 2m + k$, OMC(k) ensures that the guessed value in Step 2 for a correct transmitter is the initial value of the transmitter.*

Proof. In the case $k = 1$, the guessed value is the output of OMC(0) and it is equal to the initial value of the correct transmitter since $n - 1 > 2m$. Suppose the lemma is proved for $k - 1$. Let us prove it for k.

In OMC(k), the correct transmitter i first sends its value to the other $n - 1$ receivers. By induction hypothesis, the initial value of correct process will be guessed correctly in OMC($k - 1$). Since there are at least $n - 1 - m(> m)$ correct receivers, the decision values of all the receivers are the initial value of the transmitter. □

Lemma 12. *For any $k \geqslant 1$, OMC(k) achieves binary consensus if $n > \max\{2m + k, 2m + 2t - k\}$.*

Note that this lemma is different from Lemma 3 in the sense that $k \leqslant m$ is not required.

Proof. We proof this lemma by induction on k. The basic case $k = 1$ is the same as in Lemma 10. Hence, we suppose the lemma is proved for $k - 1$, and we prove it for k.

If the transmitter is correct, every receiver decides the same guessed value in step 2 for $n > 2m + t$. If the transmitter is faulty, then by induction hypothesis, every receiver guesses the same value for every transmitter. In the *feedback step*, the transmitter will receive $n - 1$ copies of the guessed value. Since there are at most m faulty receivers, and $n - 1 > 2m$, the transmitter also gets the same guessed value in Step 3. So the final list V_i is the same for different i proving the agreement property.

Now we consider the validity property. Suppose all input values are the same $v(\in \{0, 1\})$. Since $n > 2m + k$, by the last lemma, the guessed value for correct transmitters is the real initial value v. So there are at least $n - m(> m)$ elements in V_i with value v. Therefore, $\mathit{Major}(V_i)$ is v. The validity property is also proved. □

With this core lemma, it is easy to prove Theorem 3.

Proof. (Proof of Theorem 3) We prove the theorem by taking k equal to t in Lemma 12. □

Remark 1. *Though we have proved an upper bound for consensus, we know that this bound is not tight. For example, binary consensus is achievable with $m = 1$ and $n = 2m + t$. However, we conjecture that $2m + t$ is the tight bound if $m \geqslant n/3$.*

6 Concluding Remarks

We introduce and investigate a generalized Byzantine model inspired by component and dynamic failures. We first considered the interactive consistency problem in this model, for which we gave necessary and sufficient conditions with oral and signed messages. We also provided an upper bound for reaching binary consensus. Studying other problems in this model or applying it to the eventually synchronous context are interesting research directions.

References

[1] N. Braud-Santoni, R. Guerraoui, and F. Huc. Fast byzantine agreement. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 57–64. ACM, 2013.

[2] D. Dolev. The byzantine generals strike again. *Journal of algorithms*, 3(1):14–30, 1982.

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

[4] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. *SIAM Journal on Computing*, 17(5):975–988, 1988.

[5] M. J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *Foundations of Computation Theory*, pages 127–140. Springer, 1983.

[6] M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982.

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

[8] L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. *Dependable Computing and Fault Tolerant Systems*, 10:139–158, 1998.

[9] V. King, S. Lonargan, J. Saia, and A. Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *Distributed Computing and Networking*, pages 203–214. Springer, 2011.

[10] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[11] P. Lincoln and J. Rushby. A formally verified algorithm for interactive consistency under a hybrid fault model. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 402–411. IEEE, 1993.

[12] N. A. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.

[13] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[14] K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *Software Engineering, IEEE Transactions on*, 12(3):477–482, 1986.

[15] M. O. Rabin. Randomized byzantine generals. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 403–409. IEEE, 1983.

[16] N. Santoro and P. Widmayer. Time is not a healer. In *STACS 89*, pages 304–313. Springer, 1989.

[17] N. Santoro and P. Widmayer. Agreement in synchronous networks with ubiquitous faults. *Theoretical Computer Science*, 384(2):232–249, 2007.

[18] U. Schmid, B. Weiss, and J. Rushby. Formally verified byzantine agreement in presence of link faults. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 608–616. IEEE, 2002.

[19] T. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM (JACM)*, 34(3):626–645, 1987.

[20] S. Toueg, K. J. Perry, and T. Srikanth. Simple and efficient byzantine general algorithms with early stopping. Technical report, Cornell University, 1984.

[21] L. Tseng and N. Vaidya. Iterative approximate byzantine consensus under a generalized fault model. In *Distributed Computing and Networking*, pages 72–86. Springer, 2013.

A Omitted Proofs

A.1 Proof of Lemma 6

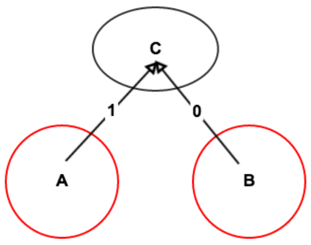

Figure 1: Case $m \geqslant t$

Proof. The case $n = 2$ is trivial. We assume $n \geqslant 3$. Suppose that F is a k-round interactive consistency algorithm. Since $n \leqslant 2m + t$, P can be partitioned into three nonempty sets A, B, and C, with $|A| \leqslant m$, $|B| \leqslant m$, $|C| \leqslant t$.

We define two scenarios α and β. In α, all processes have 0 as initial values. Processes in B are faulty and send to C processes messages pretending that processes in A have 1 as initial value.

In β, processes in A have 1 as initial value and all others processes have 0 as initial value. Processes in A are faulty and send to C messages pretending that processes in A have 0 as initial value (see Figure A.1).

In the two scenarios, processes in C get the same messages, but in the first scenario they have to decide 0 for processes in A and 1 in the second scenario leading to a contradiction.

More precisely the scenarios α and β are defined as follows:

i. For every $w \in P^+$ not starting with a process of A, let

$$\alpha(w) = \beta(w) = 0.$$

ii. For every $a \in A$, $b \in B$, $c \in C$ let

$$\alpha(a) = \alpha(aa) = \alpha(ab) = \alpha(ac) = 0,$$

$$\beta(a) = \beta(aa) = \beta(ab) = 1, \beta(ac) = 0.$$

iii. We define this part iteratively. For every $a \in A$, $b \in B$, $c \in C$, $p \in P$, $w \in aP^*$, let

$$\alpha(wcp) = \alpha(wc), \alpha(wap) = \alpha(wa),$$

$$\beta(wcp) = \beta(wc), \beta(wbp) = \beta(wb),$$

$$\alpha(wbc) = \beta(wb), \alpha(wba) = \alpha(wb),$$

$$\beta(wac) = \alpha(wa), \beta(wab) = \beta(wa).$$

α is a scenario in which processes in B are faulty and β is a scenario in which faulty processes are in set A. Moreover, $\alpha_c = \beta_c$ for all $c \in C$ then for any $a \in A$, $c \in C$

$$F_c(\alpha_c)[a] = F_c(\beta_c)[a].$$

But for any $a \in A$, $c \in C$, as F is a k-round interactive consistency algorithm

$$F_c(\alpha_c)[a] = \alpha(a) = 0,$$

$$F_c(\beta_c)[a] = \beta(a) = 1,$$

leading to a contradiction. □

A.2 Proof of Lemma 7

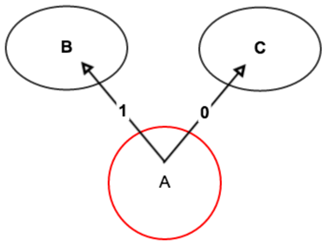

Figure 2: Case $t \geqslant m$

Proof. This proof is similar to the proof above. The case $n = 2$ is trivial. We assume $n \geqslant 3$. Suppose that F is a k-round interactive consistency algorithm. Since $n \leqslant 2t + m$, P can be partitioned into three non-empty sets A, B, and C, with $|A| \leqslant m$, $|B| \leqslant t$, $|C| \leqslant t$.

We define two scenarios α and β. In α, all processes have 0 as initial values. Processes in A are faulty and send to processes in C messages pretending that they have 1 as initial value.

In β, processes in A have 1 as initial value and all others processes have 0 as initial value. Processes in A are faulty and send to processes in B messages pretending that they have 1 as initial value (see Figure A.1).

In the two scenarios, processes in B and C get the same messages, but in the first scenario they have to decide 0 for processes in A and 1 in the second scenario giving the contradiction.

More precisely the scenarios α and β are defined as follows:

i. For every $w \in P^+$ not starting with a process of A, let

$$\alpha(w) = \beta(w) = 0.$$

ii. For every $a \in A$, $b \in B$, $c \in C$ let

$$\alpha(a) = \alpha(aa) = \alpha(ab) = 0, \alpha(ac) = 1,$$

$$\beta(a) = \beta(aa) = \beta(ac) = 1, \beta(ab) = 0.$$

iii. We define this part iteratively. For every $a \in A$, $b \in B$, $c \in C$, $p \in P$, $w \in aP^*$, let

$$\alpha(wbp) = \alpha(wb), \alpha(wcp) = \alpha(wc),$$

$$\beta(wbp) = \beta(wb), \beta(wcp) = \beta(wc),$$

$$\alpha(wab) = \alpha(wa), \beta(wac) = \beta(wa),$$

$$\alpha(wac) = \beta(wa), \beta(wab) = \alpha(wa).$$

α and β are scenarios in which processes in A are faulty. Moreover, $\alpha_b = \beta_b$, $\alpha_c = \beta_c$ for all $b \in B$ and $c \in C$. Then for any $a \in A$, $b \in B$:

$$F_b(\alpha_b)[a] = F_b(\beta_b)[a].$$

But for any $a \in A$, $b \in B$, as F is a k-round interactive consistency algorithm

$$F_b(\alpha_b)[a] = \alpha(a) = 0,$$

$$F_b(\beta_b)[a] = \beta(a) = 1,$$

leading to the contradiction. □

A.3 Proof of Lemma 9

Proof. Suppose that F is a k-round interactive consistency algorithm.

Since $n \leqslant m + 2t$, P can be partitioned into three nonempty sets A, B, and C, with $|A| \leqslant m$, $|B| \leqslant t$, $|C| \leqslant t$.

We define two scenarios α and β. In α, all processes have 0 as initial values. Processes in A are faulty and send to processes in C messages pretending that they have 1 as initial value.

In β, processes in A have 1 as initial value and all others processes have 0 as initial value. Processes in A are faulty and send to processes in B messages pretending that they have 1 as initial value.

In the two scenarios, processes in B and C get the same messages, but in the first scenario α they have to decide 0 for processes in A and 1 in the second scenario giving the contradiction.

We define scenarios α and β as follows:

i. For every $w \in P^+$ not starting with a member of A, let

$$\alpha(w) = \beta(w) = 0.$$

ii. For every $a_1 \in A^+$, $a \in A, b \in B, c \in C$, $w \in P^*$, let

$$\alpha(a) = 0, \beta(a) = 1,$$

$$\alpha(a_1bw) = \beta(a_1bw) = 0,$$

$$\alpha(a_1cw) = \beta(a_1cw) = 1.$$

iii. All other messages are sent and forwarded based on these messages correctly.

α and β are scenarios in which processes in A are faulty. Moreover, $\alpha_b = \beta_b$, $\alpha_c = \beta_c$ for all $b \in B$ and $c \in C$. Then for any $a \in A$, $b \in B$:

$$F_b(\alpha_b)[a] = F_b(\beta_b)[a]$$

But for any $a \in A$, $b \in B$, as F is a k-round interactive consistency algorithm

$$F_b(\alpha_b)[a] = \alpha(a) = 0,$$

$$F_b(\beta_b)[a] = \beta(a) = 1,$$

leading to a contradiction. $\square$