Doctoral Dissertation

A study on a trade-off between efficiency and reliability in weakly Byzantine gathering algorithms for mobile agents

Jion Hirose
Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

Supervisor: Michiko Inoue
Dependable System Lab. (Division of Information Science)

Submitted on January 30, 2024

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Jion Hirose

Thesis Committee:

Supervisor	Michiko Inoue
	(Professor, Division of Information Science)
	Shoji Kasahara
	(Professor, Division of Information Science)
	Fukuhito Ooshita
	(Adjunct Professor, Division of Information Science)
	Junya Nakamura
	(Associate Professor, Toyohashi University of Technology)
	Ryota Eguchi
	(Assistant Professor, Division of Information Science)

A study on a trade-off between efficiency and reliability in weakly Byzantine gathering algorithms for mobile agents*

Jion Hirose

Abstract

In recent years, distributed systems, comprising many interconnected computers, have become explosively popular. These systems require fault tolerance and speed enhancement to maintain efficient operation, even when some computers fail. As one of the systems satisfying this requirement, systems using mobile agents have emerged. The agents are software programs that autonomously move among nodes, and they can operate the system efficiently by cooperating. One of the most studied cooperating behaviors is gathering to make the agents, which are initially placed at arbitrary locations in the system and can only communicate with other agents at the same node, meet at a single node and declare the termination at the same time. After all agents gather, they can efficiently communicate and coordinate for future tasks. In this context, one major challenge is the gathering in a situation where some agents may fail. This is especially significant when considering the impact of agent faults on the overall functionality and efficiency of the system.

This dissertation focuses on the gathering in the presence of Byzantine agents, each of which causes Byzantine faults. Byzantine faults are known as the most severe among the various faults of agents because Byzantine agents behave maliciously. As an algorithm tolerates Byzantine agents, the existing fastest algorithm can tolerate any number of Byzantine agents, but it still requires significant time

*Doctoral Dissertation, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD2121032, January 30, 2024.

complexity. This dissertation explores the possibility of reducing time complexity by assuming that non-Byzantine agents are the majority in the system, given that, in practice, the incidence of agent faults is not typically high. We provide two efficient algorithms for scenarios with $O(f^2)$ and $O(f)$ non-Byzantine agents, where f is the number of Byzantine agents. These algorithms create groups comprising a sufficient number of non-Byzantine agents, utilizing these groups to reduce time to achieve gathering. Additionally, the second algorithm saves on the number of non-Byzantine agents by using a new technique to reach a consensus on the collected information. To reach consensus, the agents simulate a Byzantine consensus algorithm for synchronous message-passing systems on agent systems. From these results, trade-offs between reliability and efficiency in gathering problems are indicated.

Keywords:

Distributed algorithms, Fault tolerance, Mobile agents, Gathering, Byzantine environments

Contents

List of Figures

List of Tables

Part I

Introduction

1. Background

In recent years, distributed systems, comprising many interconnected computers, have become explosively popular. These systems have become integral to various applications, from the Internet to the Internet of Things (IoT). Their effectiveness stems from the capability to perform complicated tasks distributed over several nodes. However, this decentralized nature also brings challenges in terms of fault tolerance and speed enhancement. These systems require fault tolerance and speed enhancement to maintain efficient operation, even when some computers fail. Within this domain, systems using mobile agents, where these agents are software programs autonomously moving among nodes, have emerged as a solution satisfying these requirements. Their ability to independently explore the network and perform tasks collaboratively makes them valuable in enhancing system resilience and operational efficiency. Therefore, algorithms for achieving the behaviors have been studied, and prominent algorithms among these are exploration (i.e., visiting all nodes in the system), gathering (i.e., meeting at a single node), gossiping (i.e., sharing information among all agents), and so on. Researchers explore the solvability of these behaviors in various situations, and the necessary costs (time and memory required) when they are possible [14].

This dissertation considers the behavior *gathering*. This process involves agents, which are initially scattered throughout the system and are limited to communication with only others at the same node, meeting at a single node and declaring the termination. The case where precisely two agents gather is called a *rendezvous*. After all agents gather, they can effectively communicate and coordinate for future tasks. Thus, this behavior has been extensively studied in various scenarios and models [1, 30]. One critical challenge in this context is gathering in situations where some agents may fail. These faults can occur for various reasons, such as network disruptions, hardware malfunctions, software errors, or cracking. This is especially significant when considering the impact of agent faults on the

Table 1. A summary of the gathering algorithms for agents with unique IDs in synchronous environments with weakly Byzantine agents.

	Input	Condition of #Byzantine agents	Time complexity	Space complexity
Dieudonné et al. [11]	n	$f + 1 \leq k$	$O(n^4 \cdot \Lambda_{good} \cdot X(n))$	Poly. of n & Λ_{all}
Dieudonné et al. [11]	F	$2F + 2 \leq k$	Poly. of n & Λ_{good}	Poly. of n & Λ_{all}
Algorithm 1	N	$4f^2 + 9f + 4 \leq k$	$O((f + \Lambda_{all}) \cdot X(N))$	$O(k \cdot \Lambda_{all} + \log(X(N)))$
Algorithm 2	N	$8f + 7 \leq k$	$O(f \cdot \Lambda_{all} \cdot X(N))$	$O(k \cdot (\Lambda_{all} + \log(X(N)))$ $+MS_{REN}(N, 2^{\Lambda_{good}})$ $+MS_{PCONS}(S)$

overall functionality and efficiency of the system.

This dissertation focuses on the gathering in the presence of Byzantine agents, each of which causes Byzantine faults. Byzantine faults are known as the most severe among the various faults of agents because Byzantine agents behave maliciously. Therefore, we can guarantee the gathering even if the agents experience any agent faults. As algorithms tolerate Byzantine agents, several algorithms have already been proposed. Among them, the fastest can tolerate any number of Byzantine agents, but it still requires significant time complexity. The dissertation proposes solutions aimed at reducing time complexity by assuming that non-Byzantine agents are the majority in the system. While the existing algorithms are highly effective in systems prone to agent faults, they can be excessively fault-tolerant in systems with less likelihood of agent faults, leading to inefficiencies. By proposing these solutions, we can provide algorithms that are appropriately tailored to the fault tolerance of the systems. This ensures task completion efficiency.

2. Overview of This Dissertation

This dissertation aims to provide efficient algorithms that solve a gathering problem with simultaneous termination in synchronous environments, where k agents exist in the system and include some weakly Byzantine agents behaving arbitrarily, except for changing their IDs. These k agents have unique IDs but not the capability to leave information at any node, f of them are weakly Byzantine agents. Each agent initially placed at an arbitrary node wakes up at a different time and behaves in synchronous rounds. This problem requires that all agents meet at a single node and declare the termination at the same time.

Dieudonné et al. [11] proposed the fastest algorithm in this context. If all agents know the number n of nodes, this algorithm tolerates any number of weakly Byzantine agents and achieves the gathering with simultaneous termination in $O(n^4 \cdot \Lambda_{good} \cdot X(n))$ rounds, where Λ_{good} is the length of the largest ID among non-Byzantine agents, and $X(n)$ is the time required for exploring any n-nodes network (e.g., $O(n)$ rounds for a cycle, a clique, and a tree [18] and $O(n^5 \log n)$ rounds for an arbitrary graph [31]). Some papers [25, 33] proposed faster algorithms by setting additional assumptions. Miller et al. [25] assume that each agent can monitor the status of other nodes and Tsuchida et al. [33] assume that each node has dedicated memory space for each agent. However, the behaviors of Byzantine agents are significantly limited under these assumptions, which means these assumptions are strong.

In this dissertation, we assume that the number of Byzantine agents is small and propose two faster gathering algorithms with different conditions on the number of non-Byzantine agents. This assumption is considered reasonable because it is unlikely for many agents to fail in practice. Table 1 summarizes our algorithms and the fastest existing algorithm in this context. In this table, input is the information initially given to all agents; N is the upper bound on n; Λ_{all} is the length of the largest ID among agents; and F is the upper bound of f. The space complexity represents the amount of memory space required for an agent to achieve the gathering, and this is measured in bits. Variable $MS_{REN}(n, label)$ is the space complexity when an agent executes a rendezvous procedure calculated from n and a natural number *label*, and $MS_{PCONS}(S)$ is the space complexity when an agent executes a parallel Byzantine consensus algorithm calculated from an ID set S. The first algorithm assumes that $O(f^2)$ non-Byzantine agents exist and all agents know the upper bound N on n, and it achieves the gathering with simultaneous termination in $O((f + \Lambda_{all}) \cdot X(N))$ rounds, where f is the number of Byzantine agents, and Λ_{all} is the length of the largest ID among agents. The second algorithm assumes that $O(f)$ non-Byzantine agents exist and all agents know N, and it achieves the gathering with simultaneous termination in $O(f \cdot \Lambda_{all} \cdot X(N))$

rounds. Both algorithms achieve the gathering with simultaneous termination, similar to existing algorithms, which offer various benefits (e.g., all non-Byzantine agents can move to the next step after confirming that all non-Byzantine agents have met at a single node). By proposing these algorithms, trade-offs between the ratio of non-Byzantine agents to Byzantine agents and the time complexity in gathering problems in the presence of Byzantine agents are indicated.

3. Related Works

The gathering problem has been extensively studied, and most of these studies concentrate on the rendezvous scenario. These studies have the gathering problem in various environments that combine assumptions such as agent synchronization, anonymity, the presence or absence of memory on a node (called whiteboard), utilization of randomization, and network topology. These studies aim to examine the solvability of the gathering problem in these environments and measure the costs required to solve this problem, such as time, the number of moves, and memory space, if solvable. Table 2 summarizes some of the results. Pelc [30] conducted a comprehensive survey of deterministic gathering problems in various environments. Similarly, Alpern et al. [1] have reviewed randomized gathering problems in various environments. This dissertation focuses on deterministic gathering in graphs. The rest of this section details the existing results.

Many papers for the rendezvous problem assume that agents behave in synchronous rounds and whiteboards do not exist. In this scenario, agents can recognize other agents at the same node in a round, but they cannot notice other agents passing through the same edge in a round. If agents are anonymous, meaning they lack IDs, no algorithm can solve this problem in general graphs, as it is impossible to break the symmetry in some graphs such as rings. Several papers [9, 19, 32, 24] tackled this problem by assigning unique IDs to agents and revealed the time complexity in general graphs. Dessmark et al. [9] proposed the rendezvous algorithm in $\tilde{O}(n^5\sqrt{\tau\lambda} + n^{10}\lambda)$ rounds, where n is the number of nodes; λ is the length of the smallest ID among agents; τ is the maximum difference between the starting times of two agents; and $\tilde{O}$ hides a poly-logarithmic factor of n. Kowalski et al. [19] and Ta-Shma et al. [32] provided algorithms in

$\tilde{O}(n^{15}+\lambda^3)$ rounds and $\tilde{O}(n^5\lambda)$ rounds, respectively, improving the time complexity to be independent of τ. Molla et al. [26] showed a trade-off between the total number k of agents and the time complexity in the gathering problem. Their proposed algorithm works in $O(n^3)$ if $k \geq \lfloor n/2 \rfloor + 1$, in $\tilde{O}(n^4)$ if $k \geq \lfloor n/3 \rfloor + 1$, and in $\tilde{O}(n^5)$ otherwise when agents start an algorithm simultaneously and have unique IDs whose range is $[1, n^b]$ for a constant $b > 1$. Bouchard et al. [4] proposed an algorithm that solves the gathering problem when agents can detect the number of agents at the current node, but they cannot exchange any message with other agents. Miller et al. [24] investigated the trade-off between the time and the number of moves required to solve the rendezvous problem. In contrast, some studies assumed that agents are anonymous and examined the amount of memory required to solve this problem in trees [15, 7, 16] and arbitrary graphs [6]. Additionally, Dieudonneé et al. [10] investigated the gathering problem when agents are anonymous.

Several papers [23, 8, 12, 20] assumed a scenario in which agents move at different constant speeds from each other or move asynchronously. In these papers, agents are assigned unique IDs. Several papers (e.g., Kranakis et al. [20]) studied the former scenario when agents can meet other agents at nodes or inside edges, but it is unknown whether this problem can be solved in this scenario restricted to agents meeting only at nodes. In the asynchronous scenario, each agent can decide which node to visit next, but the adversary varies the movement speed of every agent arbitrarily. When agents meet other agents only at nodes, this problem in the asynchronous scenario cannot be solved in very simple graphs; thus, the papers for this problem in the asynchronous scenario assumed that agents can meet other agents at nodes or inside edges. De Marco et al. [23] clarified the costs required to solve the rendezvous problem in infinite and finite graphs. They assumed that agents initially know an upper bound of nodes in finite graphs. Czyzowicz et al. [8] removed this knowledge assumption. Dieudonné et al. [12] proposed the rendezvous algorithm in a polynomial number of moves with the number of nodes in finite graphs and the length of the smallest ID.

Recently, several papers considered the gathering problem under conditions where agent faults may occur. In the following, we discuss studies about the gathering problem in the context of crash faults, transient faults, and Byzantine faults.

Pelc [29] explored the gathering problem with crash faults, distinguishing between motion faults, where some agents stop at any time but retain memory access, and total faults, where they lose all memory upon stopping. The gathering problems in both cases require all non-faulty agents to meet at a single node because faulty agents can not gather. He proved that it is impossible to solve the gathering problem with both types of crash faults in asynchronous environments, and thus assumed that every agent moves at a fixed speed determined by the adversary and is ticking at the same rate. He proposed an algorithm achieving the gathering for the motion fault if at least two agents exist, and an algorithm achieving the gathering for the total fault if at least two non-faulty agents exist. These algorithms work in time polynomials in n, the length of the largest ID, the inverse of the smallest speed, and the ratio between the largest and the smallest speed.

Some papers [5, 27] focused on the rendezvous problem with transient faults in synchronous environments. Chalopin et al. [5] addressed delay faults that cause an agent to remain at the current node in a round r, regardless of its planned behavior in round r. If the agent planned to move in round r, it becomes aware of the delay fault in round r. The authors consider three types of delay faults: random, unbounded adversarial, and bounded adversarial. Random fault occurs independently with a constant probability $0 < p < 1$, independent for each agent in every round. In the unbounded adversarial scenario, the adversary can defer the behavior of an agent for any finite sequence of rounds, whereas in the bounded adversarial scenario, the adversary can delay the progress of an agent for at most c consecutive rounds, where c is a positive integer unknown to the agents. They measured the cost of a rendezvous algorithm through edge traversals. They proposed a rendezvous algorithm for random fault with cost polynomial in n and the length of the larger ID L in any network, succeeding with probability at least $1 - 1/n$. They showed that it is impossible to solve the rendezvous problem with unbounded adversarial in any ring, but provided the rendezvous algorithm for unbounded adversarial with cost $O(n\ell)$ in any tree, where ℓ is the smaller ID. They designed the rendezvous algorithm for bounded adversarial with cost polynomial in n, and logarithmic in c and in L in any network. Ooshita et al. [27] proposed a self-stabilizing rendezvous algorithm in any network. Self-stabilization, introduced by Dijkstra [13], is an important concept of fault tolerance in distributed systems. A self-stabilizing rendezvous algorithm ensures that if the agents start with any memory states at any two nodes, they eventually meet at the same node in the same round. Thus, this algorithm tolerates any kind of transient fault that corrupts agent memories. They provided a self-stabilizing rendezvous algorithm in any network without a time guarantee, one in any tree in time polynomial in n and ℓ, and one in any ring in time polynomial in n and their IDs.

Several papers [11, 2, 3, 33, 34, 25] considered the gathering problem in the synchronous scenario where k agents with unique IDs exist, and f of them are Byzantine agents. In this case, this problem requires all non-Byzantine agents to meet at a single node because Byzantine agents do not follow algorithms. These studies considered two types of Byzantine agents: weakly Byzantine and strongly Byzantine ones. Weakly Byzantine agents can act unpredictably except for changing their IDs, while strongly Byzantine agents may even falsify their IDs.

Dieudonné et al. [11] introduced this gathering problem under both Byzantine conditions and examined the minimum number of non-Byzantine agents ensuring the gathering in any n-nodes graphs. They proposed two gathering algorithms for weakly Byzantine agents and two for strongly Byzantine agents, all with different initial knowledge. Furthermore, they showed lower bounds on the number of non-Byzantine agents for all combinations of Byzantine conditions and initial knowledge. The first algorithm works in $O(n^4 \cdot \Lambda_{good} \cdot X(n))$ if agents know n and $k \geq f + 1$ holds, where Λ_{good} is the length of the largest ID among non-Byzantine agents, and $X(n)$ is the time required for exploring any network of n nodes. The second algorithm works in time polynomial in n and Λ_{good} if agents know the upper bound F on f and $k \geq 2F + 2$ holds. These algorithms achieved optimality regarding the required number of non-Byzantine agents. The third algorithm works in time exponential of n and Λ_{good} if agents know n and F and $k \geq 3F + 1$ holds. The fourth algorithm works in time exponential of n and Λ_{good} if agents know F and $k \geq 5F + 2$ holds. By contrast, the lower bounds of the numbers of non-Byzantine agents required to solve the gathering problem under these initial knowledge conditions are $F+1$ and $F+2$. Bouchard et al. [2] proposed two gathering algorithms with the numbers of non-Byzantine agents that match these lower bounds. Bouchard et al. [3] improved the time complexity to time polynomial in N and λ_{good} if agents know $\lceil \log \log N \rceil$ and $k \geq 5f^2 + 7f + 2$ holds.

Some papers improved the time complexity of the gathering problem with Byzantine agents using additional assumptions. Miller et al. [25] assumed that each agent can monitor the status of other agents and $k \geq 2f + 1$ holds, and they give an algorithm with $O(kn^2)$ for strongly Byzantine agents. Tsuchida et al. [33] assumed that each node is equipped with an authenticated whiteboard, where each agent can leave information on its dedicated area of the whiteboard and can read all information on the whiteboard. Their algorithm achieves the gathering in the presence of weakly Byzantine agents with $O(Fm)$ rounds if agents know F, where m is the number of edges. Additionally, using authenticated whiteboards, Tsuchida et al. [34] first proposed the algorithm that achieves the gathering problem in asynchronous scenarios with weakly Byzantine agents.

Table 2. A summary of the gathering algorithms for agents with unique IDs in synchronous environments.

	Input	Agent fault	Condition of #total agents	Time complexity
Dessmark et al. [9]	Absence	Absence	Absence	$\tilde{O}(n^5\sqrt{\tau\lambda} + n^{10}\lambda)$
Kowalski et al. [19]	Absence	Absence	Absence	$\tilde{O}(n^{15} + \lambda^3)$
Ta-Shma et al. [32]	Absence	Absence	Absence	$\tilde{O}(n^5\lambda)$
Molla et al. [26][1]	Absence	Absence	Absence	$\tilde{O}(n^5)$
	Absence	Absence	$k \geq \lfloor n/3 \rfloor + 1$	$\tilde{O}(n^4)$
	Absence	Absence	$k \geq \lfloor n/2 \rfloor + 1$	$O(n^3)$
Pelc[29][2]	Absence	Crash (Motion)	$k \geq f_c + 1$	Polynomial in $n, \Lambda_{all}, 1/\epsilon$ and γ
	Absence	Crash (Total)	$k \geq f_c + 2$	Polynomial in $n, \Lambda_{all}, 1/\epsilon$ and γ
Dieudonné et al. [11]	n	Weakly Byzantine	$k \geq f + 1$	$O(n^4 \cdot \Lambda_{good} \cdot X(n))$
	F	Weakly Byzantine	$k \geq 2F + 2$	Polynomial in n and Λ_{good}
	n, F	Strongly Byzantine	$k \geq 3F + 1$	Exponential in n and Λ_{good}
	F	Strongly Byzantine	$k \geq 5F + 2$	Exponential in n and Λ_{good}
Bouchard et al. [2]	n, F	Strongly Byzantine	$k \geq 2F + 1$	Exponential in n and Λ_{good}
	F	Strongly Byzantine	$k \geq 2F + 2$	Exponential in n and Λ_{good}
Bouchard et al. [3]	$\lceil \log \log N \rceil$	Strongly Byzantine	$k \geq 5f^2 + 7f + 2$	Polynomial in N and λ_{good}

n is the number of nodes, k is the total number of agents, f_c is the number of crash agents, f is the number of Byzantine agents, F is the upper bound of f, λ is the length of the smallest ID among agents, λ_{good} is the length of the smallest ID among non-Byzantine agents, Λ_{good} is the length of the largest ID among non-Byzantine agents, Λ_{all} is the length of the largest ID among agents, τ is the maximum difference between the starting times of two agents, ϵ is the smallest speed, γ is the ratio between the largest and the smallest speed, $X(n)$ is the time required for exploring any network of n nodes, $\tilde{O}$ hides a poly-logarithmic factor of n.

[1] They assume that all agents start at the same time and a range of agent IDs is $[1, n^b]$ for a constant $b > 1$.
[2] They assume that every agent moves at a fixed speed determined by the adversary.

4. Organization of This Dissertation

This dissertation consists of five parts. Part II introduces a formal definition for the agents model and gathering problem. Part III shows building blocks to design our proposed algorithms. Part IV and V focus on the gathering problem with different numbers of non-Byzantine agents. In Part IV, we propose a faster algorithm than [11] by relaxing the number of Byzantine agents to about the square root of the total number of agents. In Part V, we propose a faster algorithm than [11] using a linear number of non-Byzantine agents relative to the number of Byzantine agents. Finally, we discuss the time improvements and extensions to the proposed algorithms in Part VI and conclude this dissertation in Part VII.

Part II

Model and Problem Definitions

1. Model

1.1 Agent system

An agent system can be represented as a connected, undirected graph $G = (V, E)$, where V is a set of n nodes, and E is a set of edges. Nodes have no ID and lack computational, storage, or inter-node communication capabilities. The degree of a node $v \in V$ is denoted as $d(v)$. Unique port numbers in the range $\{1, \ldots, d(v)\}$ are assigned to each edge incident to node v. Notably, port numbering is local, meaning the port number assigned to an edge $e = (u, v) \in E$ at node u is independent from its number at at node v.

1.2 Agents

The agent system includes k agents, and a set of these agents is denoted by MA. Each agent $a_i \in MA$ possesses a distinct ID, $a_i.id \in \mathbb{N}$. In this dissertation, for the sake of simplicity, we use the term "an agent in an ID set" as shorthand for "an agent with an ID in an ID set S." All agents are equipped with a finite memory capacity but are incapable of leaving any information at nodes or inside edges. We model an agent as a state machine $(S, \delta, s_{ini}, S_{ter})$. Here, S is a set of agent states, δ is a state transition function, s_{ini} is an initial state, and $S_{ter} \subset S$ is a set of terminal states. Each state encapsulates a tuple representing all variable values of an agent. An agent a_i starts from s_{ini}, and it stops movement or state updating upon entering any state in S_{ter}. Agents know the upper bound N on n and their own IDs, but do not know n, k, other agent IDs, or the topology of the graph.

Agents may start from different nodes. An agent before starting is called a dormant agent, and once it starts, it is called an active agent. A dormant agent becomes active in either of the two following ways: (a) the adversary wakes up the agent at some round, or (b) an active agent visits the node with the agent.

At least one agent starting by (a) always follows an algorithm. All agents move to an adjacent node in synchronous rounds. In each round, every agent $a_i \in MA$ executes the three following operations:

Look Agent a_i discerns its state, the degree $d(v)$ of its current node u, and the port number p_u^{in} when entering node u (if u is the starting node, it notices this fact). In the presence of multiple agents at node u, a_i also identifies the IDs and the states of all agents (including those in the terminate state) at node u. We define $A_i \subseteq MA$ as a set of agents at node u, always including a_i.

Compute Agent a_i applies function δ with N and the information obtained in the last Look operation. Inputs for δ concretely include N, $a_i.id$, its state, degree $d(u)$, port number p_u^{in}, and the IDs and the states of all agents in A_i. This function determines its next state, its decision to stay or depart, and the outgoing port number p_u^{out} if departing.

Move Agent a_i either stays at node u until the start of the next round or departs from node u through port number p_u^{out}, reaching its destination node before the next round begins.

Note that if two agents traverse the same edge simultaneously in different directions, the agents do not notice this fact.

1.3 Byzantine agents

Set MA includes f weakly Byzantine agents. Weakly Byzantine agents behave unpredictably, except for changing their original IDs. When several agents are at the same node as Byzantine agents, each of them perceives identical states of the Byzantine agents in the Look operation. We denote good agents as all agents that are not weakly Byzantine agents and represent the number of good agents as $g = k - f$. Neither the actual number f of Byzantine agents nor the upper bound on f is given to good agents.

2. Problem

The gathering problem requires that all good agents reach their terminal state at a single node. In this dissertation, we categorize the gathering problem into two types based on the timing of agents entering their terminal state. The first type is the gathering problem with non-simultaneous termination, where agents can reach the terminal state in different rounds. The second type is the gathering problem with simultaneous termination, which requires all agents to enter their terminal state in the same round. To evaluate the time complexity, we count the number of rounds from the start of the earliest good agent to the point where all good agents enter their terminal state. This counting aims to establish the worst-case time complexity, considering all possible input scenarios.

Part III

Preliminaries

In this part, we give existing procedures used as building blocks to design our algorithms proposed in Parts IV and V.

1. Exploration Procedure

The exploration procedure `EX` enables an agent to visit every node in a connected graph comprising at most N nodes, starting from any node, if agents know N. This procedure stems from a corollary of the result of Reingold [31] and works according to universal exploration sequences (UXS). The total number of moves of `EX` is $O(N^5 \log N)$, which we denote as $t_{\mathtt{EX}}$. We describe the t-th round in this procedure as $\mathtt{EX}(t)$ for any integer $t \geq 0$. According to Corollary 5.5 from [31], the space complexity needed for an agent to execute `EX` is $O(\log N)$ bits.

2. Extended Label

Consider the binary representation of an agent id $a_i.id$ as $b_1 b_2 \cdots b_\ell$, where $\ell = \lfloor \log a_i.id \rfloor + 1$. We define the extended label of a_i as $a_i.id^* = 10b_1b_1b_2b_2\cdots b_\ell b_\ell 10b_1b_1b_2b_2\cdots b_\ell b_\ell \cdots$. We have the following lemma for this extended label.

Lemma **1** (Theorem 2.1 of [9]). *Assuming that two distinct agents a_i and a_j, let their extended labels $a_i.id^* = x_1x_2\cdots$ and $a_j.id^*$, respectively. It holds that $x_k \neq y_k$ for some $k \leq 2\lfloor \log(\min(a_i.id, a_j.id)) \rfloor + 6$.*

3. Rendezvous Procedure

The rendezvous procedure $\mathtt{REN}(\mathit{label})$ enables two distinct agents to meet at a single node in any connected graph comprising n nodes, where *label* is a nature number given as input. This procedure is due to Ta-Shma et al.[32]. This procedure determines the next behavior of an agent using $label$. Concretely, during the procedure, an agent alternates between exploring and waiting periods based on $label$. In the exploring period, the agent explores the network using UXS, and in the waiting period, it stays at the current node for a duration based on $label$. Meeting the two agents occurs when one agent is exploring while the other waits. The execution time of $\texttt{REN}(label)$, denoted as $t_{REN}(label)$, is $\tilde{O}(n^5 \log(label))$, where $\tilde{O}$ hides a poly-logarithmic factor in n. We observe that for two distinct inputs $label_1$ and $label_2$, it holds that $t_{REN}(label_1) > t_{REN}(label_2)$ if it holds that $label_1 > label_2$. We have the following lemma for this procedure.

Lemma **2** (Theorems 2.3 and 3.2 of Ta-Shma and Zwick [32]). *Consider two distinct agents a_i and a_j using natural numbers $label_i$ and $label_j$, respectively. If a_i starts $\texttt{REN}(label_i)$ in round r_i, a_j starts $\texttt{REN}(label_j)$ in round r_j, and $label_i \neq label_j$ holds, then they meet at a single node before round $\max(r_i, r_j) + t_{REN}(\min(label_i, label_j))$. Moreover, a_i and a_j visit every node by round $r_i + t_{REN}(label_i)$ and round $r_j + t_{REN}(label_j)$, respectively.*

We describe the t-th round in this procedure as $\texttt{REN}(label)(t)$ for any integer $t \geq 0$. We denote the space complexity needed to execute this procedure as $MS_{REN}(N, label)$, which is polynomial in n and $label$.

4. Parallel Consensus Algorithm in Byzantine Synchronous Message-Passing Systems

Our algorithm proposed in Part V employs the parallel consensus algorithm [17] for Byzantine consensus message-passing systems by simulating its mechanism for use in agent systems. This section provides an overview of the model and the characteristics of the consensus algorithm, and outlines the prerequisites for its implementation in agent systems.

4.1 Model

The message-passing distributed system, where processors exchange information via transmission, is represented as an undirected complete graph consisting of m

nodes. Each node in this system is assigned a unique ID. The system comprises at most b Byzantine nodes, which behave unpredictably except for changing their own IDs. We refer to nodes that are not Byzantine as good nodes. At the start of a process, each node knows only its own ID, and does not know m, b, or the IDs of other nodes. This system operates synchronously, namely, each node repeats synchronous phases. During a phase p, each good node conducts a local computation, transmits messages to selected nodes, and receives messages set to it in the same phase. A good node v has two options for message delivery: (1) broadcasting a message *msg* to all nodes, or (2) sending *msg* directly to a specific node whose ID is known to node v. Node v can send distinct messages to different nodes in a single phase if it knows their IDs. Each message is tagged with the sender ID, enabling the recipient to identify the sender. Byzantine nodes are unrestricted in their actions, except they cannot misrepresent their IDs to nodes that they directly communicate with.

4.2 Parallel Byzantine Consensus Problem

Each good node v possesses a set S_v consisting of k_v input pairs (id_v^i, x_v^i) for $1 \leq i \leq k_v$, where id_v^i is an ID, and x_v^i is the input value. Given that every good node v starts with S_v as its initial data, the parallel Byzantine consensus problem requires each node to output a set of pairs adhering to the following conditions:

Validity 1 The output set of a good node must include (id, x) if every good node has (id, x) as an input pair and $x \neq \perp$.

Validity 2 The output set of any good node must exclude (id, x) if it is absent as an input pair in all good nodes.

Agreement If the output set of one good node contains (id, x), then (id, x) must also be included in the output sets of all other good nodes.

Termination Each good node is required to produce a set of pairs just once, within finite phases.

This set of four conditions is called the PBC property. We say an algorithm satisfies the PBC property if the algorithm satisfies this condition set. The PBC property permits scenarios where (id, x), included in the input sets of only some but not all good nodes, might not appear in the output set of any good node.

4.3 Parallel Byzantine Consensus

Our algorithm proposed in Part V uses the parallel Byzantine consensus algorithm [17]. We denote $\texttt{PCONS}(S)$ as this algorithm for an input set S. We have the following lemma for $\texttt{PCONS}(S)$.

Lemma **3** ([17])**.** *If a system contains more than $3b$ nodes and each good node v starts $\texttt{PCONS}(S_v)$ at the same time with its input set S_v, then the execution adheres to the PBC property. In such a scenario, every good node outputs a set in $O(b)$ phases and the time to output varies by at most one phase across all good nodes.*

For any integer $p \geq 0$, We describe the action in the p-th phase of this procedure as $\texttt{PCONS}(S)(p)$. We denote the space complexity needed to execute this procedure as $MS_{PCONS}(S)$, which is polynomial in the length of the maximum ID among messages, the maximum length among messages, and the number of types of messages across the network.

4.4 Requirement for Simulation

To simulate PCONS, our algorithm proposed in Part V requires forming a group comprising many agents at least equal to the number of nodes needed for PCONS execution. Additionally, the proposed algorithm requires sending and broadcasting messages within a phase via an agent. Specifically, the algorithm needs to satisfy the following requirements:

Requirement (1) If an agent a_i in a group generates a message *msg* during a phase p, then all other good agents in the group must receive *msg* and identify $a_i.id$ before starting their local computation in the next phase $p+1$.

Requirement (2) Any agent, during a phase p, must disregard messages of a phase other than p.

Requirement (3) The group must consist of no fewer than $3f + 1$ agents.

Part IV

Gathering despite $O(\sqrt{k})$ Byzantine Agents

1. Introduction

In this part, we consider both gathering problems with non-simultaneous termination and simultaneous termination in synchronous environments with $O(\sqrt{k})$ Byzantine agents.

In this setting, the fastest existing algorithm is one proposed by Dieudonné et al. [11]. This algorithm tolerates any number of weakly Byzantine agents and achieves the gathering with simultaneous termination if agents know n; however, its time complexity is $O(n^4 \cdot \Lambda_{good} \cdot X(n))$ rounds, which is not insignificant, where Λ_{good} is the length of the largest ID among good agents, and $X(n)$ is the time required to visit all nodes of any n-nodes graph. Miller et al. [25] and Tsuchida et al. [34] proposed faster algorithms with the additional assumptions that agents can see the status of other nodes and nodes are equipped with authenticated whiteboards, respectively; however, these assumptions are strong.

We reduce the time complexity by assuming that Byzantine agents constitute few numbers. It is unlikely that many agents are subject to faults in practice; thus, this assumption is reasonable. We propose two faster algorithms if the network includes $4f^2 + 8f + 4$ good agents for f Byzantine agents and agents know N. The first algorithm achieves the gathering with non-simultaneous termination in $O((f + \Lambda_{good}) \cdot X(N))$ rounds. The second algorithm achieves the gathering with simultaneous termination in $O((f + \Lambda_{all}) \cdot X(N))$ rounds, where Λ_{all} is the length of the largest ID among agents. If n is given to agents and $\Lambda_{all} = O(\Lambda_{good})$ rounds, the second algorithm significantly reduces the time complexity compared to that [11].

2. Byzantine Gathering Algorithm with Non-Simultaneous Termination

This section shows an algorithm for the gathering problem with non-simultaneous termination by assuming that the network includes $4f^2+9f+4$ agents, that is, at least $(4f+4)(f+1)$ good agents. Recall that agents know N, but do not know n, k, or f.

2.1 Overview

The proposed algorithm aims for all good agents to gather at a single node. This objective is achieved through three stages: CollectID, MakeGroup, and Gather stages. In the CollectID stage, agents collect IDs of all good agents throughout this stage and estimate the number of Byzantine agents at the end of this stage. In the MakeGroup stage, agents make a *reliable group* comprising at least $4f+4$ agents. In the Gather stage, all good agents meet at a single node. Each stage comprises multiple phases, and each phase encompasses at least $len_p \geq t_{\mathtt{EX}}$ rounds. The exact value of len_p will be detailed later, but it should be noted that the length of each phase is sufficient for an agent to explore the network by `EX`. To simplify the explanation, we first assume that agents know f and awake at the same round. Under this assumption, all good agents start each phase at the same round.

In the CollectID stage, agents collect IDs of all good agents. To do this, in the x-th phase of the CollectID stage, each agent a_i reads the x-th bit of $a_i.id^*$ to decide its next behavior. Specifically, if the bit is 1, a_i executes `EX` in the phase. If the bit is 0, a_i waits in the phase. Agent a_i has variable $a_i.L$ to store a set of collected IDs. If a_i finds another agent a_j at the same node, either while exploring or waiting, it records $a_j.id$ in $a_i.L$. Agent a_i executes this procedure until the $(2\lfloor\log(a_i.id)\rfloor+6)$-th phase, and then finishes the CollectID stage. From Lemma 1, this stage ensures that a_i meets the other good agents; thus, a_i has the IDs of all good agents when completing this stage.

In the MakeGroup stage, agents make a reliable group comprising at least $4f+4$ agents. To do this, agents with small IDs keep waiting, while the other agents search for the agents with small IDs. Specifically, if the $f+1$ smallest IDs in $a_i.L$ contains $a_i.id$, a_i keeps waiting throughout this stage. Conversely, if these smallest IDs do not contain $a_i.id$, a_i stores the smallest ID in $a_i.L$ to variable $a_i.target$, and searches for the agent with ID $a_i.target$, say a_{target}, by executing EX in a phase. If a_i meets a_{target} at some node, it ends the search and waits there. If a_i does not meet a_{target} even after completing EX, it regards a_{target} as a Byzantine agent; then, a_i stores the second smallest ID in $a_i.L$ to $a_i.target$ and searches for the agent with ID $a_i.target$ in the next phase. This search continues until a_i meets a target agent. Given the presence of f Byzantine agents, the good agent with the smallest ID, say a_{min}, always keeps waiting during the MakeGroup stage. Thus, when agents search for a_{min}, they can meet a_{min}. Consequently, the number of agents sought by good agents is limited to at most $f+1$ (including a_{min} and f Byzantine agents). With the presence of at least $(4f+4)(f+1)$ good agents, the pigeonhole principle ensures that even if these good agents are evenly distributed to $f+1$ nodes, at least $4f+4$ agents meet at some node. In other words, agents can make a reliable group. The ID of the found target agent becomes the ID of this reliable group. For Gather stage, this reliable group is divided into two groups: an exploring group and a waiting group, each comprising at least $2f+2$ agents.

In the Gather stage, agents achieve the gathering after at least one reliable group is created. The Gather stage consists of two phases. In the first phase, agents collect group IDs of all reliable groups. More concretely, while agents in a waiting group keep waiting, the others (in an exploring group or not in a reliable group) explore the network by EX. Upon meeting a reliable group, an agent a_i records the group ID. It should be noted that both an exploring group and a waiting group contain at least $2f+2$ agents each when it is created. This means each group contains at least $f+2$ good agents. Therefore, when an agent meets a group including at least $f+2$ agents with the same group ID, the agent can understand that this group contains at least two good agents; hence, the group is trustworthy. As we explain later, the algorithm makes each group include at least two good agents because the agent must use estimated values of f and the estimated values of f differ by at most one among good agents. In the second phase, agents move to the node with the waiting group of the smallest group ID. In other words, while agents in the waiting group of the smallest group ID keep waiting, other agents search for the group by EX.

However, implementing the above behavior presents three problems.

The first problem arises from the fact that agents not in a reliable group cannot immediately know the formation of a reliable group. This uncertainty leaves the agents unclear about the time to transition into the Gather stage. To address this, we make agents execute the MakeGroup stage and the Gather stage alternately. We design the two stages to satisfy the following conditions: (1) If a reliable group is created in the MakeGroup stage, agents achieve the gathering in the Gather stage; (2) Otherwise, the behaviors in the Gather stage do not affect the MakeGroup stage.

The second problem is that agents do not know f. To solve this problem, agents estimate the number of Byzantine agents, say $\tilde{f}$, at the end of the CollectID stage. The agents base this estimation on the fact that their ID sets include IDs of all good agents, at least $(4f+4)(f+1)$ good agents exist, and the number of Byzantine agents is at most f. Specifically, good agents determines $\tilde{f}$ such that it satisfies $k \geq (4\tilde{f}+4)(\tilde{f}+1)$ and $f \leq \tilde{f}$. This approach ensures the formation of at least one reliable group. However, their $\tilde{f}$ differs by at most one because some good agents may meet some Byzantine agents but the others may not in the CollectID stage. To counter this variance, we design a technique for creating a reliable group such that both an exploring group and a waiting group include at least $\tilde{f}'+1$ good agents, where $\tilde{f}'$ is the largest value of $\tilde{f}$ among all good agents.

The third problem is that some agents may be dormant. To solve this problem, we make agents first explore the network by EX to wake up dormant agents. This ensures that all good agents start within t_{EX} rounds. However, a new problem arises as agents, having awakened at different times, are likely to be in different phases at some round. To resolve this, we adjust the duration of each phase to guarantee that all the good agents execute the same phase at the same time for sufficient rounds.

2.2 Details

Algorithm 1 is the pseudocode of the proposed algorithm. The proposed algorithm realizes the gathering using three stages: The CollectID stage makes

Algorithm 1: `ByzantineGathering`(N) for an agent a_i whose ID is $b_1 b_2 \cdots b_\ell$, where $\ell = \lfloor \log(a_i.id) \rfloor + 1$

```
1  a_i.state ← CorrectID
2  a_i.L ← {a_i.id}, a_i.BL ← ∅, a_i.GL ← ∅
3  a_i.gid ← NULL
4  a_i.EndCI ← False
5  a_i.x ← 1
7  Explore the network by EX
8  while True do
9      if a_i.EndCI = False then
10         Execute a_i.x-th phase of the CollectID stage
11     else
12         Execute the MakeGroup stage
13     a_i.x ← a_i.x + 1
14     Execute the Gather stage
```

agents collect IDs of all good agents and estimate the number of Byzantine agents, the MakeGroup stage creates a reliable group, and the Gather stage gathers all good agents.

The overall flow of the algorithm is shown in Fig. 1. After starting the algorithm, agent a_i first explores the network with EX to wake up all dormant agents (line 7 of Algorithm 1). By this behavior, after the first good agent wakes up, all good agents wake up within t_{EX} rounds. After that, a_i executes phases of the CollectID, MakeGroup, and Gather stages. Here we define one phase as $len_p = 3t_{EX} + 1$ rounds. Since all good agents wake up within t_{EX} rounds, the $(t_{EX} + 1)$-th to $2t_{EX}$-th rounds of the x-th phase of good agent a_i overlap with the first $3t_{EX}$ rounds of the x-th phases of all other good agents. Hence, we have the following observation.

Observation **1.** *Let a_i and a_j be good agents. Assume that a_i explores the network with EX from the $(t_{EX} + 1)$-th round to the $2t_{EX}$-th round of its x-th phase, and a_j waits during the first $3t_{EX}$ rounds of its x-th phase. In this case, a_i meets a_j during the exploration.*

After the initial exploration, a_i alternately executes one phase of the CollectID stage and two phases of the Gather stage (lines 10 and 14). Because a_i cannot calculate the value of $\tilde{f}$ until it finishes the CollectID stage, a_i takes no action in the Gather stage. After a_i finishes the CollectID stage, it alter-

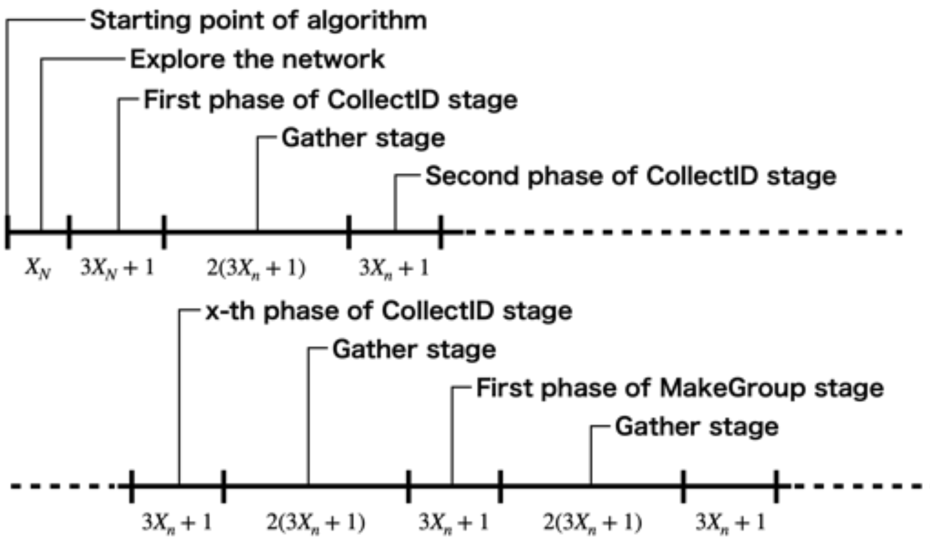

Figure 1. Stage flow of Algorithm `ByzantineGathering`.

nately executes one phase of the MakeGroup stage (instead of the CollectID stage) and two phases of the Gather stage (lines 12 and 14). The Gather stage interrupts the CollectID and MakeGroup stages, but, as described later, the behaviors of the Gather stage do not affect the behaviors of the CollectID and MakeGroup stages if no reliable group exists. Therefore, we do not consider the behaviors of the Gather stage until a reliable group is created in the MakeGroup stage.

An example execution of the algorithm by some good agents a_i, a_j, and a_k is shown in Fig. 2. Capitals C, M, and G represent one phase of the CollectID stage, one phase of the MakeGroup stage, and two phases of the Gather stage, respectively. Recall that agents need to execute multiple phases of the CollectID stage (resp., the MakeGroup stage) to achieve the purpose of the CollectID stage (resp., the MakeGroup stage) and that agents alternately execute one phase of the CollectID stage (resp., the MakeGroup stage) and two phases of the Gather stage. Let r_1 be the round when a reliable group with a_i is created, and r_2 be the round when a_k finished the CollectID stage. Agents a_i and a_j terminate at the end of the Gather stage immediately after round r_1 since they have finished the CollectID stage and a reliable group exists in the network. On the other hand, agent a_k cannot determine whether a

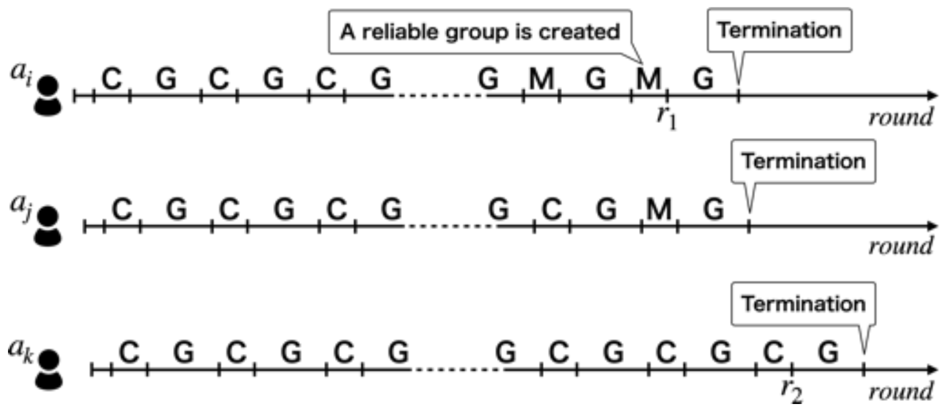

Figure 2. Example execution of the algorithm by good agents a_i, a_j, and a_k.

reliable group exists at the same node since it has not finished the CollectID stage. Thus, agent a_k keeps executing. Afterward, agent a_k terminates at the end of the Gather stage immediately after round r_2 because a reliable group already exists in the network and a_k meets the reliable group by the end of the Gather stage.

Table 3 summarizes the variables used in the algorithm. Agent a_i stores the current state of a_i in variable $a_i.state$. Initially, $a_i.state = CorrectID$ holds. Initially, a_i stores $False$ in variable $a_i.EndCI$ because it has not finished the CollectID stage. Also, a_i stores the number of rounds from the beginning in variable $a_i.count$. By variable $a_i.count$, a_i determines which round of a phase it executes, and so, when a_i waits, it can obtain how many rounds it has waited for. Agent a_i increments $a_i.count$ for every round, but this behavior is omitted from the following description.

2.2.1 CollectID stage

Algorithm 2 is the pseudocode of the CollectID stage. In the CollectID stage, agents collect IDs of all good agents. The CollectID stage of a_i consists of $2\lfloor\log(a_i.id)\rfloor + 6$ phases. Note that the lengths of CollectID stages differ among agents. Agent a_i uses variable $a_i.L$ to store a set of IDs, and initially, it records $a_i.id$ in $a_i.L$ (line 2 of Algorithm 1). Agent a_i determines the behavior of the x-th phase depending on the x-th bit of $a_i.id^*$. If the x-th bit is 0, a_i waits for $3t_{EX}$ rounds in the x-th phase (lines 1 and 2 of Algorithm 2). If the x-th bit

Table 3. Variables of agent a_i.

Variable	Explanation
state	The current state of an agent. This variable takes one of the following values: • *CorrectID* (has not yet finished the CollectID stage) • *SearchAgent* (works as a search agent in the MakeGroup stage) • *TargetAgent* (works as a target agent in the MakeGroup stage) • *ExploringGroup* (belongs to an exploring group in the Gather stage) • *WaitingGroup* (belongs to a waiting group in the Gather stage)
EndCI	The variable that indicates whether an agent has finished the CollectID stage
count	The number of rounds from the beginning
x	The number of phases in the CollectID or MakeGroup stage
$\tilde{f}$	The estimated number of Byzantine agents
L	A set of agent IDs collected in the CollectID stage
BL	A set of agent IDs that the search agent regards as Byzantine agents
target	Search agents: The ID the agent searches for. Target agents: Its own ID
F	The consensus of $\tilde{f}$ among agents at the same node
gid	The group ID of the reliable group that the agent belongs to
GL	A set of group IDs collected in the Gather stage

is 1, a_i waits for t_{EX} rounds, explores the network by EX, and then waits for t_{EX} rounds in the x-th phase (lines 4–7). During these behaviors, if a_i finds another agent a_j at the same node, it records $a_j.id$ in $a_i.L$ (lines 3 and 8). Note that, from Lemma 1 and Observation 1, a_i meets all good agents and records IDs of all good agents during the CollectID stage.

In the last round of the last phase of the CollectID stage, a_i estimates the number of Byzantine agents $\tilde{f}$ as $a_i.\tilde{f} \leftarrow \max\{y \mid (4y+4)(y+1) \leq |a_i.L|\}$ (line 11). As we prove later, $a_i.\tilde{f} \geq f$ holds in Lemmas 5 and 6, and $|a_i.\tilde{f} - a_j.\tilde{f}| \leq 1$ holds for any good agent a_j. Also, a_i stores $True$ in $a_i.EndCI$ (line 13).

2.2.2 MakeGroup stage

Algorithm 3 is the pseudocode of the MakeGroup stage. In the pseudo code, for simplicity we use **and** operation, which means that an agent executes the operations before and after the **and** operation at the same time.

In the MakeGroup stage, agents create a reliable group. To store a group

Algorithm 2: The $a_i.x$-th phase of CollectID stage for an agent a_i

if *the $a_i.x$-th bit of $a_i.id^*$ is 0* **then**
  Wait for $3t_{EX}$ rounds at the current node
  $a_i.L \leftarrow a_i.L \cup \{$IDs of agents a_i met while waiting$\}$
else
  Wait for t_{EX} rounds at the current node
  Explore the network by `EX`
  Wait for t_{EX} rounds at the current node
  $a_i.L \leftarrow a_i.L \cup \{$IDs of agents a_i met while exploring$\}$
// The $(3t_{EX} + 1)$-th round
if $a_i.x = 2\lfloor \log(a_i.id) \rfloor + 6$ **then**
  $a_i.\tilde{f} \leftarrow \max\{y \mid (4y + 4)(y + 1) \le |a_i.L|\}$
  $a_i.x \leftarrow 1$
  $a_i.EndCI \leftarrow True$
Wait for one round

ID of the reliable group, agent a_i has variable $a_i.gid$. Initially $a_i.gid$ is $NULL$, and, when a_i becomes a member of a reliable group, it assigns its group ID to $a_i.gid$. Let $\tilde{f}_{min}$ be the smallest value of $\tilde{f}$ among all good agents at the time when all good agents finish the CollectID stage. We define a reliable group formally to present the MakeGroup stage clearly.

Definition **1** (Reliable group)**.** *A set of agents R is a reliable group with group ID gid if and only if R includes at least $3\tilde{f}_{min} + 4$ good agents and $a_i.gid = gid$ holds for any $a_i \in R$*

At the beginning of the MakeGroup stage, if the smallest $a_i.\tilde{f}+1$ IDs in $a_i.L$ contain $a_i.id$, agent a_i becomes a *target agent* (line 3 of Algorithm 3). Otherwise, a_i becomes a *search agent* (line 5). Hereinafter, the good agent with the smallest ID is denoted by a_{min}. As we prove later, a_{min} always becomes a target agent.

If a_i is a target agent, it executes $a_i.target \leftarrow a_i.id$ (line 8) and waits for one phase at the current node (line 9). While waiting, a_i executes procedure `consensus()` to create a reliable group if possible (line 11). We will explain the details of `consensus()` later.

Let us consider the case where a_i is a search agent. Here, to ensure making a reliable group, a_i stores IDs of agents that a_i regards as Byzantine agents in the blacklist $a_i.BL$ (initially $a_i.BL$ is empty). In the first round of each phase, a_i chooses the agent with the smallest ID, excluding Byzantine agents in $a_i.BL$

Algorithm 3: MAKEGROUP stage for an agent a_i

1 **if** $a_i.x = 1$ **then**
2   **if** *the smallest* $a_i.\tilde{f} + 1$ *IDs in* $a_i.L$ *contain* $a_i.id$ **then**
3         $a_i.state \leftarrow TargetAgent$
4   **else**
5         $a_i.state \leftarrow SearchAgent$
6 **if** $a_i.state = TargetAgent$ **then**
7     $//a_i$ is a target agent
8     $a_i.target \leftarrow a_i.id$
9   Wait for one phase at the current node
10   **and**
11   While waiting, execute `consensus()` every round
12 **else**
13     $//a_i$ is a search agent
14     $a_i.target \leftarrow \min(a_i.L \setminus a_i.BL)$
15   Wait for t_{EX} rounds at the current node
16   Search for an agent a_{target} with ID $a_i.target$ by `EX`
17   **and**
18   **if** a_i *meets* a_{target} *while searching* **then**
19   Stop `EX`
20   Wait until the end of the phase
21   **and**
22   While waiting, execute `consensus()` every round
23   **and**
24   **if** a_i *finds* a_{target} *Byzantine while waiting* **then**
25   // This is true if, during the $(t_{EX} + 1)$-th round to the $2t_{EX}$-th round, a_{target} moved to another node or $a_{target}.target \neq a_{target}.id$ holds
26             $a_i.BL \leftarrow a_i.BL \cup \{a_i.target\}$
27   **else**
28   // a_i does not meet a_{target} and hence a_{target} is Byzantine
29         $a_i.BL \leftarrow a_i.BL \cup \{a_i.target\}$
30   Wait until the end of the phase

(line 14). After that, a_i waits for t_{EX} rounds and then searches for the agent with ID $a_i.target$, say a_{target}, by executing `EX` (lines 15 and 16). If a_i finds a_{target} at the same node during the exploration, a_i ends `EX` and waits at the node until the end of the phase (lines 19 and 15). We can show that, if a_{target} is good, a_{target} keeps waiting as a target agent, and consequently, a_i finds a_{target} and waits with a_{target}. Hence, if one of the following conditions holds, a_i regards a_{target} as a Byzantine agent: (1) a_i did not find a_{target} during the exploration (lines 27–29), or (2) after a_i found a_{target}, during the $(t_{EX} + 1)$-th round to the $2t_{EX}$-th round, a_{target} moved to another node or $a_{target}.target \neq a_{target}.id$ holds (lines 24–26). In this case, a_i

Algorithm 4: `consensus()` for an agent a_i (Compute the consensus of $\tilde{f}$ and create a reliable group if possible)

1 **if** *$a_i.gid$ = NULL and the number of agents in the* MakeGroup *stage at the current node is at least* $4 \cdot a_i.\tilde{f}$ **then**
2     $a_i.F \leftarrow$ the most frequent value of $\tilde{f}$ of agents at the same node (if more than one most frequent value exists, choose the smallest one)
3   Let GC be a set of agents at the same node whose *target* is $a_i.target$ and who execute the MakeGroup stage
4   **if** $|GC| \geq 4 \cdot a_i.F + 4$ *and there exists* a_{target} *with* $a_{target}.target = a_{target}.id = a_i.target$ **then**
5         $a_i.gid \leftarrow a_{target}.id$
6   **if** *the* $2 \cdot a_i.F + 2$ *smallest IDs in GC contain* $a_i.id$ **then**
7             $a_i.state \leftarrow ExploringGroup$
8   **else**
9             $a_i.state \leftarrow WaitingGroup$

adds $a_{target}.id$ to $a_i.BL$, and never searches for a_{target} in the later phases of the MakeGroup stage (lines 26 and 29). If a_i did not find a_{target}, it waits until the end of the phase (line 30).

To determine whether agents can create a reliable group, search agents (resp., target agents) execute procedure `consensus()` in Algorithm 4 after they find their target agent (resp., from the beginning). In procedure `consensus()`, agent a_i first calculates the consensus $a_i.F$ of the estimated number of Byzantine agents as follows. If the number of agents in the MakeGroup stage at the current node is at least $4 \cdot a_i.\tilde{f}$, agent a_i checks values of $\tilde{f}$ of all agents at the current node and assigns the most frequent value to $a_i.F$ (line 2 of Algorithm 4). At this time, if multiple values are the most frequent, a_i chooses the smallest one.

After that, a_i determines whether the agent can create a reliable group. Agent a_i observes states of all agents at the same node, and regards the set of agents whose *target* is $a_i.target$ and who execute the MakeGroup stage as the *group candidate* (line 3). If the group candidate contains at least $4 \cdot a_i.F + 4$ agents and there exists a_{target} with $a_{target}.target = a_{target}.id = a_i.target$, a_i recognizes that the group candidate includes $3\tilde{f}_{min} + 4$ good agents since $a_i.F \geq \tilde{f}_{min} \geq f$ holds (Lemmas 5 and 9) (line 4). In that case, a_i is ready to make a reliable group. Agent a_i regards the group candidate as a reliable group and stores $a_{target}.id$ in variable $a_i.gid$ as the group ID of the reliable group (line 5). Note that, as we prove later, all other good agents in the reliable group also understand that they are in the reliable group and assign $a_{target}.id$ to their variable *gid* at the same round. Therefore, when a_i assigns a group ID *gid* to $a_i.gid$, a reliable group with *gid* is indeed created. If a_i meets another agent a_j, a_i can identify whether a_j is a member of a reliable group by observing variable $a_j.gid$. When a reliable group is created, the group is divided into two groups, an *exploring group* and a *waiting group*, for the GATHER stage as follows. If the $2 \cdot a_i.F + 2$ smallest IDs among agents in a_i's reliable group contain $a_i.id$, a_i belongs to an exploring group (line 7); otherwise, it belongs to a waiting group (line 9). Note that each of an exploring group and a waiting group contains at least $2 \cdot a_i.F + 2 \geq 2\tilde{f}_{min} + 2$ agents. Because $\tilde{f}$ of every good agent that finished the COLLECTID stage satisfies $\tilde{f} \leq \tilde{f}_{min} + 1$, all good agents understand that these groups contain at least one good agent (Lemma 6). Hence, these groups are trustworthy.

Once a_i has determined that a reliable group is created, it never calculates $a_i.F$ and checks the condition to create a reliable group again in subsequent rounds of this phase. Note that some good agent a_j with $a_j.target = a_{target}.id$ may visit the current node after a_i creates a reliable group. In this case, a_j can become a member of the reliable group (i.e., $a_j.gid \leftarrow a_{target}.id = a_i.gid$). This just increases the size of the reliable group and does not harm the algorithm.

2.2.3 GATHER stage

Algorithm 5 is the pseudocode of the GATHER stage. In the GATHER stage, agents achieve the gathering if at least one reliable group exists in the network. Note that two phases of the GATHER stage interrupt phases of the COLLECTID and MAKEGROUP stages. However, while executing the GATHER stage, agents never update variables used in the COLLECTID and MAKEGROUP stages. Also, recall that the behaviors of the COLLECTID and MAKEGROUP stages do not depend on the initial positions of agents in each phase. Hence, the behaviors of the GATHER stage do not affect the behaviors of the COLLECTID and MAKEGROUP stages. If agents have not finished the COLLECTID stage, they wait for two phases (lines 1 and 2 of Algorithm 5). In the following, we describe the behaviors of agents that have finished the COLLECTID stage.

If agents have finished the COLLECTID stage, they try to achieve the gathering

Algorithm 5: Gather stage for an agent a_i

1 **if** $a_i.EndCI = False$ **then**
2   Wait for two phases at the current node
3 **else**
4   // The first phase
5   **if** $a_i.state = WaitingGroup$ **then**
6   Wait for one phase at the current node
7   **and**
8   While waiting, whenever a_i meets a_j with $a_j.gid \neq NULL$, execute $a_i.GL \leftarrow a_i.GL \cup \{(a_j.gid, a_j.id)\}$
9   **else**
10   Wait for t_{EX} rounds at the current node
11   Explore the network by `EX`
12   **and**
13   While exploring, whenever a_i meets a_j with $a_j.gid \neq NULL$, execute $a_i.GL \leftarrow a_i.GL \cup \{(a_j.gid, a_j.id)\}$
14   Wait for $t_{EX} + 1$ rounds at the current node
15   // The second phase
16     $//MemberID(gid) = \{id \mid (gid, id) \in a_i.GL\}$
17     $//ReliableGID() = \{gid \mid |MemberID(gid)| \geq a_i.\tilde{f} + 1\}$
18   **if** $ReliableGID() = \varnothing$ **then**
19   Wait for one phase at the current node
20   **else if** $a_i.state = WaitingGroup$ *and* $a_i.gid = \min(ReliableGID())$ **then**
21   Wait for $3t_{EX}$ rounds at the current node
22   Terminate the algorithm
23   **else**
24   Wait for t_{EX} rounds at the current node
25   By executing `EX`, search for the node with a reliable waiting group whose group ID is $\min(ReliableGID())$
26   Wait at the node until the last round of the phase
27   Terminate the algorithm at the last round of the phase

in two phases of the Gather stage. In the first phase of the two phases, agents collect group IDs of all reliable groups (lines 4–14). To do this, agents in waiting groups keep waiting for the phase, and other agents (agents in exploring groups and agents not in reliable groups) explore the network during the $(t_{EX} + 1)$-th round to the $2t_{EX}$-th round. During this behavior, when an agent finds a waiting or exploring group, it records the group ID. After that, in the second phase, they gather at the node where the reliable group with the smallest group ID exists (lines 15–27).

Here, we explain how agents find exploring or waiting groups. Since agents enter the Gather stage at different rounds, agents in a reliable group do not move together. This implies that agent a_i meets agents in a reliable group at different rounds. For this reason, whenever agent a_i meets a_j with $a_j.gid \neq NULL$ (i.e., a_j says it is in a reliable group), a_i adds a pair $(a_j.gid, a_j.id)$ in a set $a_i.GL$. Then, at the beginning of the second phase, a_i checks $a_i.GL$ and computes group IDs of reliable groups. More concretely, a_i determines that gid is a group ID of a reliable group if there exist at least $a_i.\tilde{f}+1$ different IDs $id_1, id_2, \ldots$ such that $(gid, id_k) \in a_i.GL$ for any k, that is, the number of agents that conveyed gid as their group IDs is at least $a_i.\tilde{f}+1$. In the rest of this paragraph, we explain why this threshold $a_i.\tilde{f}+1$ allows agent a_i to recognize a reliable group correctly. Assume that agent a_i finds the exploring or waiting group of a reliable group. Recall that the exploring or waiting group initially contains at least $2\tilde{f}_{min}+2$ agents. From this fact, even if $f \leq \tilde{f}_{min}$ of them are Byzantine, at least $\tilde{f}_{min}+2$ good agents convey their group ID to a_i. Consequently, when a_i finds the group, a_i can determine that at least one good agent exists in this group because $|a_i.\tilde{f} - \tilde{f}_{min}| \leq 1$ holds (as shown in Lemmas 6 and 9). Therefore, if a_i finds an exploring or waiting group (i.e., agents with the same gid) composed of at least $a_i.\tilde{f}+1$ agents, a_i can correctly recognize the group as an exploring or waiting group of a reliable group.

In the following, we explain the detailed behavior of agent a_i in the two continuous phases of the GATHER stage.

In the first phase, to collect all group IDs, agents in waiting groups keep waiting, and other agents (agents in exploring groups and agents not in reliable groups) explore the network. To be more precise, if agent a_i belongs to a waiting group, a_i collects pairs of a group ID and an agent ID in variable $a_i.GL$ by waiting and observing visiting agents. That is, a_i waits for one phase, and if a_i finds agent a_j with $a_j.gid \neq NULL$ while waiting, it adds $(a_j.gid, a_j.id)$ to $a_i.GL$ (lines 6–8). If agent a_i belongs to a exploring group or does not belong to a reliable group, a_i collects pairs of a group ID and an agent ID in variable $a_i.GL$ by exploring the network. That is, a_i waits for t_{EX} rounds, explores the network, and then waits for $t_{EX}+1$ rounds. If a_i finds agent a_j with $a_j.gid \neq NULL$ during the exploration, it adds $(a_j.gid, a_j.id)$ to $a_i.GL$ (lines 10–14).

In the second phase, all agents gather at the node where the reliable group with the smallest group ID exists. Initially, a_i calculates the set $ReliableGID()$ of group IDs of all reliable groups as follows: (1) a_i makes, for each group ID *gid* in $a_i.GL$, a list of agent IDs that conveyed *gid* as its group ID (i.e., $\mathit{MemberID}(gid) = \{id \mid (gid, id) \in a_i.GL\}$), and (2) a_i checks up group IDs such that at least $a_i.\tilde{f} + 1$ agents conveyed the group ID (i.e., $\mathit{ReliableGID}() = \{gid \mid |\mathit{MemberID}(gid)| \geq a_i.\tilde{f} + 1\}$). Note that, if a_i belongs to a exploring (resp., waiting) group, $a_i.gid \in \mathit{ReliableGID}()$ holds because a_i meets members of its own waiting (resp., exploring) group during the first phase. If a_i belongs to a waiting group and satisfies $a_i.gid = \min(\mathit{ReliableGID}())$, it waits for $3t_{\mathtt{EX}}$ rounds and terminates the algorithm (lines 20–22). Otherwise, a_i waits for $t_{\mathtt{EX}}$ rounds and searches for the node with the waiting group whose group ID is $\min(\mathit{ReliableGID}())$ by executing `EX` (lines 23–25). After that, a_i waits until the last round of this phase and terminates the algorithm at the node (lines 26–27).

2.3 Correctness and Complexity

In this subsection, we prove the correctness and complexity of the proposed algorithm.

Lemma **4.** *Let a_i be a good agent. When a_i finishes the* CollectID *stage, $a_i.L$ contains IDs of all good agents.*

Proof. By Lemma 1 and Observation 1, a_i meets all good agents before the end of the CollectID stage, and records their IDs in $a_i.L$. Therefore, $a_i.L$ contains IDs of all good agents at the end of the CollectID stage. □

Lemma **5.** *After good agent a_i finishes the* CollectID *stage, $a_i.\tilde{f} \geq f$ and $k \geq (4a_i.\tilde{f} + 4)(a_i.\tilde{f} + 1)$ hold.*

Proof. By Lemma 4, a_i contains IDs of all good agents in $a_i.L$ at the end of CollectID stage, and so $|a_i.L| \geq (4f + 4)(f + 1)$ holds. Therefore, we have $a_i.\tilde{f} = \max\{y \mid (4y{+}4)(y{+}1) \leq |a_i.L|\} \geq \max\{y \mid (4y{+}4)(y{+}1) \leq (4f{+}4)(f{+}1)\} = f$. Also, by the algorithm, we clearly have $k \geq (4a_i.\tilde{f} + 4)(a_i.\tilde{f} + 1)$. □

Lemma **6.** *After good agents a_i and a_j finish the* CollectID *stage, $|a_i.\tilde{f} - a_j.\tilde{f}| \leq 1$ holds.*

Proof. We prove this lemma by contradiction. Without loss of generality, we assume $a_i.\tilde{f} = p$ and $a_j.\tilde{f} \geq p + 2$. We have $(4(p + 1) + 4)((p + 1) + 1) > |a_i.L|$

by $a_i.\tilde{f} < p+1$, and we have $(4(p+2)+4)((p+2)+1) \le |a_j.L|$ by $a_j.\tilde{f} \ge p+2$. Therefore, since $p \ge f$ holds by Lemma 5, $|a_j.L| - |a_i.L| > 8p + 20 > f$ holds. On the other hand, since $a_i.L$ and $a_j.L$ include IDs of all good agents by Lemma 4, we have $|a_j.L| - |a_i.L| \le f$, which contradicts the assumption. □

Let $\tilde{f}_{max}$ be the largest value of $\tilde{f}$ among all good agents at the time when all good agents finish the CollectID stage.

Lemma **7.** *The followings hold in the* MakeGroup *stage: (1) a_{min} is a target agent, and (2) the number of good target agents is at most $\tilde{f}_{max} + 1$.*

Proof. First, we prove proposition (1). By Lemma 5, $a_{min}.\tilde{f} \ge f$ holds; thus, the $a_{min}.\tilde{f} + 1$ ($\ge f + 1$) smallest IDs in $a_{min}.L$ contain $a_{min}.id$. Therefore, a_{min} is a target agent.

Next, we prove proposition (2) by contradiction. Let us assume that proposition (2) does not hold. That is, at least $\tilde{f}_{max} + 2$ good agents become target agents. Let a_{max} be the agent with the largest ID among the good target agents. Since $a_{max}.L$ contains IDs of other $\tilde{f}_{max} + 1$ good agents that have smaller IDs than a_{max}, a_{max} does not become a target agent. This is a contradiction. Hence, the lemma holds. □

Lemma **8.** *Let a_i be a good agent. Variable $a_i.BL$ does not contain any ID of good agents.*

Proof. We prove by induction. Recall that a_i adds $a_i.target$ to $a_i.BL$ in a phase of the MakeGroup stage only when one of the following conditions holds. Let a_{target} be the agent such that $a_i.target = a_{target}.id$ holds.

1. Agent a_i did not find a_{target} during the phase (line 29 of Alg. 3).

2. After a_i found a_{target}, during the $(t_{EX}+1)$-th round to the $2t_{EX}$-th round of the phase, a_{target} moved to another node or $a_{target}.target \ne a_{target}.id$ holds (line 26 of Alg. 3).

For the base case, we consider the first phase of the MakeGroup stage of a_i. By Lemma 4, $a_i.L$ contains IDs of all good agents. Since $a_i.BL$ is empty at the beginning of the first phase, $a_i.target$ ($= \min(a_i.L)$) is $a_{min}.id$ or an ID of a

Byzantine agent. But, here, it is sufficient to consider only the former case. Since a_{min} has the smallest ID among good agents, the duration of the CollectID stage is the shortest among good agents. Hence, a_{min} starts the MakeGroup stage before a_i starts the $(t_{EX}+1)$-th round of the first phase of the MakeGroup stage. Since a_{min} is a target agent by Lemma 7, a_{min} continues to wait during the MakeGroup stage. This implies that the above conditions to update $a_i.BL$ are not satisfied. Hence, a_i does not update $a_i.BL$, and the lemma holds in the first phase.

For the induction, assume that $a_i.BL$ does not contain IDs of good agents at the end of the t-th phase of the MakeGroup stage of a_i. We consider the $(t+1)$-th phase of the MakeGroup stage of a_i. Since $a_i.BL$ does not contain IDs of the good agents at the beginning of the $(t+1)$-th phase, $a_i.target = \min(a_i.L \setminus a_i.BL)$ is $a_{min}.id$ or an ID of a Byzantine agent. By the same discussion as in the first phase, we can prove that IDs of good agents are not added to $a_i.BL$ in the $(t+1)$-th phase. Therefore, this lemma holds in the $(t+1)$-th phase. Hence, the lemma holds. □

In the following lemmas, we show the property of a reliable group.

Lemma **9.** *When good agent a_i executes $a_i.F \leftarrow \tilde{f}'$ in* `consensus`(), *there exists good agent a_j with $a_j.\tilde{f} = \tilde{f}'$*

Proof. Assume that a_i executes $a_i.F \leftarrow \tilde{f}'$ at node v in round r. By the algorithm, in round r, there exist at least $4 \cdot a_i.\tilde{f}$ agents executing the MakeGroup stage at node v. Since $a_i.\tilde{f} \geq f$ holds by Lemma 5, there exist at least $4 \cdot a_i.\tilde{f} - f \geq 4f - f = 3f$ good agents executing the MakeGroup stage at v in round r. Also, since variable $\tilde{f}$ of good agents takes at most two possible values by Lemma 6, at least $\lceil 3f/2 \rceil > f$ good agents at v have the same value of $\tilde{f}$. Therefore, in round r, a_i stores the value of variable $\tilde{f}$ of some good agent in $a_i.F$. Hence, the lemma holds. □

Lemma **10.** *If good agent a_i executes $a_i.gid \leftarrow gid$ (line 5 of Algorithm 4) at node v in round r, (1) a reliable group with group ID gid is created in round r, and (2) an exploring and waiting group of the reliable group is created in round r and each of them contains at least $\tilde{f}_{min} + 2$ good agents.*

Proof. Assume that good agent a_i executes $a_i.gid \leftarrow gid$ at v in round r. Let A' be a set of agents such that, iff $a_j \in A'$ holds, a_j stays at v in round r and $a_j.target = a_i.target$ holds.

Firstly, we prove that A' satisfies the following conditions.

- Set A' contains at least $4 \cdot a_i.F + 4$ agents.
- Any good agent a_j in A' executes $a_j.gid \leftarrow gid$ at node v in round r.

Since a_i executes $a_i.gid \leftarrow gid$, A' contains at least $4 \cdot a_i.F + 4$ agents. Also, A' contains agent a_{target} with $a_{target}.id = a_i.target$. Fix an agent $a_j \in A'$. By Lemmas 6 and 9, $a_j.\tilde{f} \leq a_i.F + 1$ holds, and hence, $4 \cdot a_i.F + 4 \geq 4 \cdot a_j.\tilde{f}$ holds. This implies that the number of agents at v satisfies the condition that a_j calculates $a_j.F$ (line 1 of Algorithm 4). Since the situation of v is the same for both a_i and a_j, $a_j.F = a_i.F$ holds. In addition, a_j also observes agents in A'; then, a_j executes $a_j.gid \leftarrow gid$ at v in round r.

Secondly, we prove (1). By Lemmas 5, 6 and 9, A' contains at least $4 \cdot a_i.F + 4 - f \geq 4\tilde{f}_{min} + 4 - f \geq 4\tilde{f}_{min} + 4 - \tilde{f}_{min} = 3\tilde{f}_{min} + 4$ good agents. Also, for any $a_i \in A'$, $a_i.gid = gid$ holds. Therefore, A' is a reliable group with group ID *gid* from Definition 1.

Lastly, we prove (2). Each agent $a_j \in A'$ (including a_i) also decides an exploring or wating group of the reliable group with group ID *gid* at node v in round r. Since A' contains at least $4 \cdot a_i.F + 4 \geq 4\tilde{f}_{min} + 4$ agents, each of the exploring and waiting groups contains at least $2\tilde{f}_{min} + 2$ agents. Therefore, each of the exploring and waiting groups contains at least $2\tilde{f}_{min} + 2 - f \geq \tilde{f}_{min} + 2$ good agents. □

In the following two lemmas, we prove that a reliable group is created before all good agents finish the $(f+1)$-th phase of the MakeGroup stage. Let a_{last} be the good agent that finishes the CollectID stage last, and let $phase_x$ be the x-th phase of the MakeGroup stage of a_{last}. Since all agents wake up within t_{EX} rounds and each phase consists of $3t_{EX} + 1$ rounds, any good agent a_i has exactly one phase $phase_x^i$ that overlaps $phase_x$ for at least $2t_{EX} + 1$ rounds. For simplicity, when agent a_i behaves in $phase_x^i$, we say that a_i behaves in the x-th phase (of the MakeGroup stage) of a_{last}.

Lemma **11.** *Let $Byz_1, Byz_2, \ldots, Byz_{f'}$ ($Byz_l.id < Byz_{l+1}.id$ for $1 \le l \le f' - 1$) be Byzantine agents whose IDs are smaller than a_{min}. Assume that, when a_{last} finishes the f'-th phase of the* MakeGroup *stage, a reliable group does not exist. Then, in the $(f'+1)$-th phase of the* MakeGroup *stage of a_{last}, at most $(4\tilde{f}_{max} + 2)f'$ good agents assign bid $\in \{Byz_1.id, Byz_2.id, \ldots, Byz_{f'}.id\}$ to their variable target.*

Proof. Assume that a reliable group does not exist when a_{last} finishes the f'-th phase of the MakeGroup stage. Under this assumption, we prove by induction that, in the $(x+1)$-th phase of the MakeGroup stage ($1 \le x \le f'$) of a_{last}, at most $(4\tilde{f}_{max} + 2)x$ good agents assign $bid \in \{Byz_1.id, Byz_2.id, \ldots, Byz_x.id\}$ to their variable *target*. Hereinafter, the x-th phase of the MakeGroup stage of a_{last} is simply called the x-th phase.

For the base case, we consider the case of $x = 1$. Let A_1 be a set of good agents that assign $Byz_1.id$ to their variable *target* in the second phase. For contradiction, assume $|A_1| > 4\tilde{f}_{max} + 2$. Since good agents monotonically increase *target*, agents in A_1 also assign $Byz_1.id$ to *target* in the first phase. Also, since the agents do not regard Byz_1 as a Byzantine agent in the first phase, they find Byz_1 in the first phase and, after that, Byz_1 does not move and $Byz_1.target = Byz_1.id$ holds until the $2t_{EX}$-th round of the first phase. In addition, they start the first phase within at most t_{EX} round and wait during the $(2t_{EX}+1)$-th round to the $(3t_{EX}+1)$-th round of the first phase. This implies that all agents in A_1 exist at the same node as Byz_1 before the $2t_{EX}$-th round of the first phase, and at that time the number of agents at the node is at least $|A_1 \cup \{Byz_1\}| \ge 4\tilde{f}_{max} + 4$. Furthermore, since those agents have stored $Byz_1.id$ in their $target$, they assign $Byz_1.id$ to their *gid* (execute line 5 of Algorithm 4). By Lemma 10, since a reliable group is created by the algorithm, this contradicts the assumption. Therefore, $|A_1| \le 4\tilde{f}_{max} + 2$ holds.

For induction step, assume that, in the $(x+1)$-th phase ($1 \le x < f'$), at most $(4\tilde{f}_{max} + 2)x$ good agents assign $bid \in \{Byz_1.id, Byz_2.id, \ldots, Byz_x.id\}$ to their *target*. Let A_x be a set of good agents that assign $bid \in \{Byz_1.id, Byz_2.id, \ldots, Byz_{x+1}.id\}$ to *target* in the $(x+2)$-th phase. For contradiction, assume $|A_x| > (4\tilde{f}_{max} + 2)(x+1)$. Let B_x be a set of good agents that assign $Byz_{x+1}.id$ to *target* in the $(x+1)$-th phase, and let C_x be a set of good agents that assign

$bid \in \{Byz_1.id, Byz_2.id, \dots, Byz_x.id\}$ to *target* in the $(x+1)$-th phase. Since good agents monotonically increase *target*, $A_x \subseteq B_x \cup C_x$ holds. Since $|C_x| \le (4\tilde{f}_{max}+2)x$ holds by the assumption of induction, $|B_x \cap A_x| \ge |A_x| - |C_x| > 4\tilde{f}_{max} + 2$ holds. Since good agents in $B_x \cap A_x$ do not regard Byz_{x+1} as a Byzantine agent in the $(x+1)$-th phase, they find Byz_{x+1}, and, after that, Byz_{x+1} does not move and $Byz_{x+1}.target = Byz_{x+1}.id$ holds until the $2t_{\mathcal{EX}}$-th round of the $(x+1)$-th phase. Similarly to the base case, this implies that all agents in $B_x \cap A_x$ exist at the same node as Byz_{x+1}, and at that time, the number of agents at the node is at least $4\tilde{f}_{max} + 4$. Furthermore, since those agents have stored $Byz_{x+1}.id$ in their *target*, they assign $Byz_{x+1}.id$ to their *gid* (execute line 5 of Algorithm 4). By Lemma 10, since a reliable group is created by the algorithm, this contradicts the assumption. Therefore, $|A_x| \le (4\tilde{f}_{max} + 2)(x+1)$ holds.

Hence, the lemma holds. □

Lemma **12.** *Before a_{last} finishes the $(f+1)$-th phase of the* MakeGroup *stage, a reliable group is created.*

Proof. Let $f'(\le f)$ be the number of Byzantine agents whose IDs are smaller than $a_{min}.id$. By Lemma 11, if a reliable group is not created before a_{last} finishes the f'-th phase of the MakeGroup stage, at most $(4\tilde{f}_{max} + 2)f'$ good agents assign an ID of a Byzantine agent with a smaller ID than a_{min} to *target* in the $(f'+1)$-th phase of a_{last}. Also, by Lemma 7, the number of good target agents is at most $\tilde{f}_{max} + 1$. This implies that, in the $(f'+1)$-th phase of a_{last}, at least $(k-f) - (\tilde{f}_{max} + 1) - (4\tilde{f}_{max} + 2)f'$ good search agents assign $a_{min}.id$ to *target* (because $a_{min}.id$ is not in variable BL of agents by Lemma 8). Since they can successfully find a_{min}, by Lemma 5, at least $(k-f) - (\tilde{f}_{max}+1) - (4\tilde{f}_{max}+2)f' \ge (4\tilde{f}_{max}+4)(\tilde{f}_{max}+1) - \tilde{f}_{max} - (\tilde{f}_{max}+1) - (4\tilde{f}_{max}+2)\tilde{f}_{max} = 4\tilde{f}_{max}+3$ search agents stay with target agent a_{min} before the $2t_{\mathcal{EX}}$-th rounds of the $(f'+1)$-th phase of a_{last}. This implies that at least $4\tilde{f}_{max} + 4$ agents with $target = a_{min}.id$ exist at the node with a_{min}. Therefore, they assign $a_{min}.id$ to their *gid* (execute line 5 of Algorithm 4). By Lemma 10, a reliable group is created. Hence, the lemma holds. □

The following two lemmas show that agents can achieve the gathering if at least one reliable group is created and they finish the CollectID stage. Let

a_{ini} be the good agent that wakes up earliest. Since all agents wake up within t_{EX} rounds, if a_{ini} starts two consecutive phases of the GATHER stage in round r, all good agents start two consecutive phases of the GATHER stage before round $r + t_{EX}$. We define $Rel(r)$ as a set of reliable groups that exist in round $r + t_{EX}$. If $Rel(r)$ is not empty, we define $gid_{min}(r)$ as the smallest group ID of reliable groups in $Rel(r)$, $G_{min}(r)$ as the group with group ID $gid_{min}(r)$, and $v_{min}(r)$ as the node where $G_{min}(r)$ is created.

Lemma **13.** *Consider the following situation: (1) a_{ini} starts two consecutive phases of the* GATHER *stage in round r, (2) a_i (possibly a_{ini}) starts two consecutive phases of the* GATHER *stage in round r' such that $r \leq r' \leq r + t_{EX}$ holds, and (3) a_i has completed the* COLLECTID *stage before round r'. Let $List_i$ be the output of ReliableGID() for a_i in the two consecutive phases. Then, $List_i$ is a set of all group IDs of $Rel(r)$.*

Proof. By the algorithm, since all good agents wake up within t_{EX} rounds, all good agents start two consecutive phases of the GATHER stage during rounds r to $r + t_{EX}$ and hence, no new reliable group is created during rounds $r + t_{EX}$ to $r + 2t_{EX}$.

If a_i belongs to a waiting group, it waits during rounds $r'(\leq r + t_{EX})$ to $r' + 3t_{EX}(\geq r + 3t_{EX})$. Since all good agents in exploring groups of $Rel(r)$ explore the network during rounds $r + t_{EX}$ to $r + 3t_{EX}$, all of them meet a_i. Therefore, for each good agent a in a exploring group of $Rel(r)$, $a_i.GL$ contains $(a.gid, a.id)$.

If a_i does not belong to a waiting group, it explores the network during rounds $r' + t_{EX}(\geq r + t_{EX})$ to $r' + 2t_{EX}(\leq r + 3t_{EX})$. Since all good agents in waiting groups of $Rel(r)$ wait during rounds $r + t_{EX}$ to $r + 3t_{EX}$, all of them meet a_i. Therefore, for each good agent a in a waiting group of $Rel(r)$, $a_i.GL$ contains $(a.gid, a.id)$.

Let G be an arbitrary group in $Rel(r)$. By Lemma 10, each of the exploring group and the waiting group of G contains at least $\tilde{f}_{min} + 2$ good agents. By Lemma 6, since $a_i.\tilde{f} + 1 \leq \tilde{f}_{max} + 1 \leq \tilde{f}_{min} + 2$ holds, $a_i.GL$ contains at least $a_i.\tilde{f} + 1$ pairs for group G. Hence, $List_i$ contains all group IDs of $Rel(r)$. In addition, since there exist only $f < a_i.\tilde{f} + 1$ Byzantine agents, $List_i$ does not contain a fake group ID that was conveyed by Byzantine agents. Hence, $List_i$ is a set of all group IDs of $Rel(r)$. □

Lemma **14.** *Let r be the first round such that (a) a_{ini} starts two consecutive phases of the* Gather *stage in round r and (b) there exists a reliable group in round $r + t_{EX}$. Assume that a_i (possibly a_{ini}) starts two consecutive phases of the* Gather *stage in round r' such that $r \le r' \le r + t_{EX}$. Then, the following propositions hold: (1) If a_i has finished the* CollectID *stage before round r', it terminates the algorithm at $v_{min}(r)$ during the two consecutive phases of the* Gather *stage after round r'. (2) If a_i has not finished the* CollectID *stage in round r', it terminates the algorithm at $v_{min}(r)$ in the first two consecutive phases of the* Gather *stage after it finishes the* CollectID *stage.*

Proof. First, we prove proposition (1). We focus on the first two consecutive phases of the Gather stage after round r'. From Lemma 13, a_i obtains the set of all group IDs of $Rel(r)$ as the output of *ReliableGID()* and hence, min(*ReliableGID()*) is $gid_{min}(r)$. Hence, if a_i belongs to a waiting group of $G_{min}(r)$, it terminates at its current node $v_{min}(r)$ at the $(3t_{EX} + 1)$-th round of the second phase after round r'. Otherwise, a_i searches for the waiting group of $G_{min}(r)$ in the second phase after round r'. More concretely, a_i explores the network during the $(t_{EX} + 1)$-th round to the $2t_{EX}$-th round in the second phase. Recall that agents in a waiting group of $G_{min}(r)$ wait for $3t_{EX}$ rounds before terminating at $v_{min}(r)$ in their second phases, and the difference of starting times of the phases is at most t_{EX}. Hence, a_i meets agents in a waiting group of $G_{min}(r)$ at $v_{min}(r)$ during the exploration, and then, it terminates at $v_{min}(r)$.

Next, we prove proposition (2). Consider the case that a_i is the first agent that finishes the CollectID stage after r'. Assume that, in round r'', a_i finishes the CollectID stage. Since all agents that have finished the CollectID stage before round r' have terminated from proposition (1), no agent executes the MakeGroup stage between r' and r'', and so the set of reliable groups is $Rel(r)$. Since all agents that belong to groups in $Rel(r)$ have terminated from proposition (1), a_i meets all of them in the first phase of the Gather stage after round r''. Hence, in the second phase, $gid_{min}(r) = \min(\textit{ReliableGID()})$ holds, and consequently a_i terminates the algorithm at $v_{min}(r)$ during the second phase. Consider the case that a_i is not the first agent that finishes the CollectID stage after r'. Similarly to the above case, no agent executes the MakeGroup stage after r', and consequently the set of reliable groups is still $Rel(r)$. Hence, we can prove this case similarly to the above case. □

Finally, we prove the complexity of the proposed algorithm.

Theorem **1.** *Let n be the number of nodes, k be the number of agents, f be the number of weakly Byzantine agents, Λ_{good} be the largest ID among good agents, and Λ_{all} be the largest ID among agents. If the upper bound N of n is given to agents and $4f^2 + 9f + 4 \leq k$ holds, the proposed algorithm solves the gathering problem with non-simultaneous termination in at most $t_{EX} + 3(2\lfloor\log(\Lambda_{good})\rfloor + f + 7)(3t_{EX} + 1)$ rounds using $O(k \cdot \log(\Lambda_{all}) + \log X(N)) + MS_{REN}(N, \Lambda_{good})$ bits of agent memory.*

Proof. Let a_{last} be the good agent that finishes the CollectID stage last. Since a_{last} wakes up within t_{EX} rounds (after the first agent wakes up) and executes at most $2\lfloor\log(\Lambda_{good})\rfloor+6$ phases of the CollectID stage, a_{last} finishes the CollectID stage in $t_{EX}+(2\lfloor\log(\Lambda_{good})\rfloor+6)\cdot 3(3t_{EX}+1) = t_{EX}+3(2\lfloor\log(\Lambda_{good})\rfloor+6)(3t_{EX}+1)$ rounds. By Lemma 12, a reliable group is created before a_{last} finishes the $(f+1)$-th phase of the MakeGroup stage. By Lemma 14, if at least one reliable group is created and all good agents finish the CollectID stage, agents achieve the gathering during the next two phases of the Gather stage. Therefore, agents achieve the gathering in at most $t_{EX} + 3(2\lfloor\log(\Lambda_{good})\rfloor + 6)(3t_{EX} + 1) + (f + 1) \cdot 3(3t_{EX} + 1) = t_{EX} + 3(2\lfloor\log(\Lambda_{good})\rfloor + f + 7)(3t_{EX} + 1)$ rounds.

Next, we analyze the space complexity required for an agent a_i to execute `ByzantineGathering`. We first consider the amount of memory space required for a_i to keep every variable.

Case Variable *state* and *EndCI*: Agent a_i stores a constant number of parameters to $a_i.state$ and $a_i.EndCI$; thus, the amounts of memory space of these variables are $O(1)$ bits.

Case Variable *count*: The above discussion gives at most $t_{EX}+3(2\lfloor\log(\Lambda_{good})\rfloor+f+7)(3t_{EX}+1)$ as the upper bound of $a_i.count$; thus, the amount of memory space of the variable is $O(\log(f + \Lambda_{good})X(N)) = O(\log(f + \Lambda_{good}) + \log(X(N)))$ bits.

Case Variable x: Agent a_i stores $2\lfloor\log(a_i.id)\rfloor + 6$ to $a_i.x$ and the maximum ID among good agents is Λ_{good}; thus, the amount of memory space of the variable is $O(\log(\Lambda_{good}))$ bits.

Case Variables $\tilde{f}$ and F: Lemmas 5, 6, and 9 shows at most $f+1$ as the upper bounds of $a_i.\tilde{f}$ and $a_i.F$: thus, the amounts of memory space of these variables are $O(\log(f))$ bits.

Case Variables L and GL: Agent a_i stores only IDs of agents it has met; therefore, it stores at most k agents IDs to $a_i.L$ and $a_i.GL$. The maximum ID among IDs stored by a_i is Λ_{all}; thus, the mounts of memory space of these variables are $O(k\log(\Lambda_{all}))$ bits.

Case Variable BL: Lemma 8 shows that a_i stores at most f agent IDs to $a_i.BL$ and the maximum ID among IDs stored by a_i is Λ_{all}; thus, the amount of memory space of the variable is $O(f\log(\Lambda_{all}))$ bits.

Case Variables *target* and *gid*: Agent a_i stores only one ID to $a_i.target$ and $a_i.gid$ and the upper bounds on these variables are Λ_{good}; thus, the amounts of memory space of these variables are $O(\log(\Lambda_{good}))$ bits.

The amount of memory space required for a_i to keep every variable is $O(k\log(\Lambda_{all}) + \log(X(N)))$ bits. As mentioned in Section 1, the amount of memory space of the exploration procedure is $O(\log(N)$. Thus, the space complexity required for an agent to execute `ByzantineGathering` is $O(k\log(\Lambda_{all}) + \log(X(N)))$ bits. □

3. Byzantine Gathering Algorithm with Simultaneous Termination

In this section, we propose an algorithm for the gathering problem *with simultaneous termination* by modifying the algorithm in the previous section. The underlying assumption is the same as that of the previous section. In the following, we refer to the proposed algorithm in the previous section as the previous algorithm. In the previous algorithm, all good agents gather at a single node but can terminate at different rounds. Therefore, the purpose of this section is to change the termination condition of the previous algorithm so that all good agents terminate at the same round.

By Lemma 14, after all good agents finish the CollectID stage and at least one reliable group is created, all good agents gather at a single node during the next two consecutive phases of the Gather stage. Hence, after good agents move to the gathering node in the GATHER stage, they can terminate at the same round if they wait until all good agents finish the COLLECTID stage (and the next GATHER stage). To do this, we can use the fact that, when good agent a_i finishes the COLLECTID stage, $a_i.L$ contains IDs of all good agents. That is, $\max(a_i.L)$ is the upper bound of IDs of good agents and hence, a_i can compute the upper bound of rounds required for all good agents to finish the COLLECTID stage. However, for two good agents a_i and a_j, $\max(a_i.L)$ can be different from $\max(a_j.L)$ because it is possible that either a_i or a_j meets a Byzantine agent with an ID larger than the largest ID among good agents. Also, if agents share their variable L and take the maximum ID, Byzantine agents may share a very large ID such that no agent has the ID. To overcome this problem, each agent a_i selects the largest ID among IDs that $a_i.F + 1$ agents have in their variable L, and computes when to terminate. Note that, in order that all good agents agree on the largest ID, they should have the same value of F. For this reason, each agent a_i updates $a_i.F$ similarly to the MAKEGROUP stage after it completes the previous algorithm. Since all good agents in a reliable group exist at a single node, a_i can correctly update $a_i.F$.

Lastly, to terminate at the same round, good agents make a consensus on termination. To do this, each agent a_i prepares a flag $a_i.flag_t$ (initially, $a_i.flag_t \leftarrow False$). Agent a_i executes $a_i.flag_t \leftarrow True$ if it is ready to terminate, i.e., it understands that all good agents gather at the current node. After a_i completes the previous algorithm, it also checks $flag_t$ of all agents at the current node every round. If $flag_t$ of at least $a_i.F + 1$ agents are true, a_i terminates the algorithm because at least one good agent understands that all good agents gather at the current node. Since all good agents stay at the same node and make the decision based on the same information, they can terminate at the same round.

In the rest of this section, we describe the detailed behavior of a_i in the algorithm. First, a_i executes the previous algorithm until just before it terminates, but it does not terminate. Let round r_i be the round immediately after a_i completes the previous algorithm. After round r_i, a_i waits at the gathering node of the previous algorithm, say v, and always checks whether it can terminate. More concretely, a_i executes the following operations every round after round r_i.

1. Agent a_i updates $a_i.F$ in the same way as in the MAKEGROUP stage of the

previous algorithm, that is, a_i assigns the most frequent value of $\tilde{f}$ to $a_i.F$. If multiple values are the most frequent, a_i chooses the smallest one.

2. Agent a_i checks $flag_t$ of agents at v, and, if $flag_t$ of at least $a_i.F+1$ agents are true, a_i terminates the algorithm.

3. Agent a_i checks variable L of agents at v and computes the maximum ID among agents. That is, letting L_g be a set of IDs that at least $a_i.F+1$ agents at v have in their variable L, a_i executes $a_i.id_{max} \leftarrow \max(L_g)$.

4. Agent a_i checks whether all good agents gather at v. If all good agents have completed the CollectID stage before round r_i, all good agents gather at v before round $r_i + t_{EX}$ because all agents wake up within t_{EX} rounds. Consider the case that some good agent has not yet completed the CollectID stage in round r_i. Since a reliable group has already been created, if the agent with ID $a_i.id_{max}$ has finished the CollectID stage and its next two phases of the Gather stage, a_i understands that all good agents gather at v. Note that the agent with ID $a_i.id_{max}$ completes the CollectID stage and its next two phases of the Gather stage in at most $T = t_{EX} + t_{EX} + 3(2\lfloor\log(a_i.id_{max})\rfloor + 6)(3t_{EX} + 1)$ rounds after a_i starts the algorithm. For this reason, a_i sets $a_i.flag_t \leftarrow True$ if (a) t_{EX} rounds have elapsed after round r_i and (b) T rounds have elapsed after it started the algorithm.

Theorem **2.** *Let n be the number of nodes, k be the number of agents, f be the number of Byzantine agents, and Λ_{all} be the largest ID among all agents. If the upper bound N of n is given to agents and $4f^2+9f+4 \le k$ holds, the proposed algorithm solves the gathering problem with simultaneous termination in at most $3t_{EX}+3(2\lfloor\log(\Lambda_{all})\rfloor+f+7)(3t_{EX}+1)+1$ rounds using $O(k\log(\Lambda_{all})+\log(X(N)))$ bits of agent memory.*

Proof. Let a_{ini} be the agent that starts the algorithm earliest. Let r be the first round such that (a) a_{ini} starts two consecutive phases of the Gather stage in round r and (b) there exists a reliable group in round $r+t_{EX}$, and let $Rel(r)$ be a set of reliable groups that exist in round $r+t_{EX}$. Let $G_{min}(r)$ be the group with the smallest group ID in $Rel(r)$, and let $v_{min}(r)$ be the node where $G_{min}(r)$ is created. From Lemma 14, each good agent exists at $v_{min}(r)$ when it completes the previous algorithm.

Let a_f be the agent that executes $flag_t \leftarrow True$ earliest, and assume that a_f executes $a_f.flag_t \leftarrow True$ in round r^*.

First, we prove that all good agents complete the previous algorithm before round r^*. Assume that a_f completes the previous algorithm in round r_f. If all good agents complete the CollectID stage before round r_f, all good agents gather at v before round $r_f + t_{EX}$. Since $r^* \geq r_f + t_{EX}$ holds, all good agents complete the previous algorithm before round r^*. Consider the case that some good agent has not yet completed the CollectID stage in round r_f. Since all agents wake up within t_{EX} rounds and agents do not move during the last t_{EX} rounds of the previous algorithm, good agents in a reliable group in $Rel(r)$ exist at $v_{min}(r)$ after round r_f. Hence, at least $4 \cdot a_f.F + 4 - f \geq 3f$ good agents exist at $v_{min}(r)$ after round r_f. Hence, similarly to Lemma 9, a_f assigns $\tilde{f}$ of some good agent to $a_f.F$ after round r_f. This implies that a_f assigns an ID of some agent to $a_f.id_{max}$. Note that the assigned ID is at least Λ_{good}, where Λ_{good} is the largest ID among all good agents. Hence, since a_f executes $flag_t \leftarrow True$ only when T rounds have elapsed from the beginning, all good agents complete the CollectID stage and the next two consecutive phases of the Gather stage in round r^*. Since a reliable group has already been created, all good agents complete the previous algorithm before round r^*.

Next, we prove that all good agents terminate at $v_{min}(r)$ at the same round. From the above discussion, all good agents wait at $v_{min}(r)$ in round r^*. Since all good agents obtain the same information at $v_{min}(r)$, they decide the same value on F. Hence, they can terminate at the same round immediately after at least $F + 1$ agents execute $flag_t \leftarrow True$.

Next, we prove that good agents terminate in at most $3t_{EX} + 3(2\lfloor \log(\Lambda_{all}) \rfloor + f + 7)\ (3t_{EX} + 1) + 1$ rounds. Similarly to Theorem 1, all good agents complete the previous algorithm and gather at $v_{min}(r)$ in at most $T_1 = t_{EX} + 3(2\lfloor \log(\Lambda_{good}) \rfloor + f + 7)(3t_{EX} + 1)$ rounds. In addition, since id_{max} is an ID of some agent, good agents wait until at most $T_2 = 2t_{EX} + 3(2\lfloor \log(\Lambda_{all}) \rfloor + 6)(3t_{EX} + 1)$ rounds have passed. Note that good agents execute $flag_t \leftarrow True$ if (a) t_{EX} rounds have passed after they completed the previous algorithm and (b) $T (\leq T_2)$ rounds have passed after the beginning of the algorithm. Hence, good agents execute $flag_t \leftarrow True$ in at most $T_3 = max\{T_1 + t_{EX}, T_2\} \le 2t_{EX} + 3(2\lfloor\log(\Lambda_{all})\rfloor + f + 7)(3t_{EX} + 1)$ rounds after they start the algorithm. Since all good agents start the algorithm within t_{EX} rounds and they terminate after at least $F+1$ agents execute $flag_t \leftarrow True$, they terminate in at most $t_{EX} + T_3 + 1 = 3t_{EX} + 3(2\lfloor\log(\Lambda_{all})\rfloor + f + 7)(3t_{EX} + 1) + 1$ rounds after the first good agent wakes up.

Finally, we analyze the space complexity required for an agent a_i to execute the proposed algorithm. The proposed algorithm uses all variables and building blocks of the previous algorithm; hence, the space complexity is at least $O(k\log(\Lambda_{all}) + \log(X(N)))$ bits by Theorem 1. In the proposed algorithm, a_i additionally uses variables flag_t, L_g, id_{max}, *count*, r_i, and T; thus, we analyze the amount of the memory space of these variables.

Case Variable flag_t: Agent a_i stores a context number of parameters to this variable; thus, the amount of memory space of this variable is $O(1)$ bits.

Case Variable L_g: Agent a_i stores IDs that at least $a_i.F + 1$ agents at the gathered node include in their variable L. By Lemmas 5 and 9, $a_i.F \ge f$ holds; therefore, at least one good agent has these IDs in its variable L. A good agent stores only the IDs of agents it has met to L and the maximum ID in its L is at most Λ_{all}. Thus, the amount of memory space of this variable is $O(k\Lambda_{all})$ bits.

Case Variable id_{max}: Agent a_i stores one ID from $a_i.L_g$ to $a_i.id_{max}$ and the upper bound on id_{max} is Λ_{all} from above discussion; thus, the amount of memory space of this variable is $O(\Lambda_{all})$ bits.

Case Variable *count*: The discussion on time complexity gives at most $t_{EX} + T_3 + 1 = 3t_{EX} + 3(2\lfloor\log(\Lambda_{all})\rfloor + f + 7)(3t_{EX} + 1) + 1$ as the upper bound of $a_i.count$; thus, the amount of memory space of the variable is $O(\log(f + \Lambda_{all})X(N)) = O(\log(f + \Lambda_{all}) + \log(X(N)))$ bits.

Case Variables r_i and T: Theorem 1 gives $O(\log(f + \Lambda_{good}) + \log(X(N)))$ bits as the upper bound of $a_i.r_i$. The discussion on $a_i.count$ gives $O(\log(f + \Lambda_{all}) + \log(X(N)))$ bits as the upper bound of $a_i.T$.

Thus, the space complexity required for an agent to execute the proposed algorithm is $O(k\log(\Lambda_{all}) + \log(X(N)))$ bits. □

4. Summary

In this part, we proposed gathering algorithms with different termination characteristics in the presence of $O(\sqrt{k})$ Byzantine agents. These algorithms reduced the time complexity by assuming that the network includes many agents. More specifically, if N is given to agents, and at least $(4f + 4)(f + 1)$ exist in the network, the first algorithm achieves the gathering with non-simultaneous termination in $O((f + \Lambda_{good}) \cdot X(N))$ rounds, and the second algorithm achieves the gathering with simultaneous termination in $O((f + \Lambda_{all}) \cdot X(N))$ rounds. In these algorithms, several good agents first create a reliable group such that good agents can trust the behavior of the group to suppress the influence of Byzantine agents. Subsequently, the reliable group collects the other good agents, and all good agents gather at a single node. To create a reliable group, good agents with the smallest ID in the collected IDs wait and other good agents search for the waiting agents.

Part V

Gathering despite $O(k)$ Byzantine Agents

1. Introduction

In this part, we consider both gathering problems with non-simultaneous termination and simultaneous termination in synchronous environments with $O(k)$ Byzantine agents.

Dieudonné et al. [11] researched to clarify the minimum number of good agents required to solve the gathering problem with simultaneous termination. As a result, they proposed an algorithm tolerating any number of Byzantine agents; however, its time complexity is $O(n^4 \cdot \Lambda_{good} \cdot X(n))$ rounds, which is not insignificant, where Λ_{good} is the length of the largest ID among good agents, and $X(n)$ is the time required to visit all nodes of any n-nodes graph. The study in the previous part assumes that the network includes a few Byzantine agents and investigates whether this assumption could shorten the time required to solve these problems. As a result, we propose the gathering algorithm with simultaneous termination in $O((f + \Lambda_{all}) \cdot X(N))$ rounds, which is the fastest in the context, where Λ_{all} is the length of the largest ID among agents; however, this algorithm requires at least $4f^2 + 8f + 4$ good agents, which is not a small number. In summary, the first algorithm requires a small number of good agents, but has high time complexity, while the second algorithm has low time complexity, but can only $o(k)$ number of Byzantine agents. Thus, no existing algorithms have both low time complexity and a small number of good agents.

We propose two gathering algorithms with different termination characteristics and low time complexity in the presence of $\Omega(f)$ good agents. If agents know N and the network includes at least $8f + 7$ agents, the first algorithm achieves the gathering with non-simultaneous termination in $O(f \cdot \Lambda_{good} \cdot X(N))$ rounds and the second algorithm achieves the gathering with simultaneous termination in $O(f \cdot \Lambda_{all} \cdot X(N))$ rounds. If n is given to agents, the second algorithm is faster than that [11] and requires fewer good agents than the one in the previous part. To solve the gathering problems under these assumptions, we propose herein a new technique of simulating a consensus algorithm [17] for synchronous Byzantine message-passing systems on agent systems, in which one agent simulates one process of the message-passing system. Byzantine consensus is solvable on a synchronous distributed system with at least $3b + 1$ processes, where b is the number of Byzantine processes [28, 21]. However, it is difficult for all agents to simulate synchronous rounds of Byzantine message-passing systems and start the consensus algorithm at the same time. We instead construct a group of at least $3f + 1$ agents that simulate the algorithm. The proposed technique is a universal technique for simulating an algorithm for message-passing systems on agent systems.

2. Byzantine Gathering Algorithm with Non-Simultaneous Termination

In this section, we first present an overview of the proposed algorithm. Next, we provide the details of the proposed algorithm, including the explanation of a sub-algorithm to design the proposed algorithm. Throughout the paper, we assume $k = g + f \geq 8f + 7$, which implies that there are at least $7f + 7$ good agents in the network. Recall that agents know N, but do not know n, k, or f.

2.1 Overview

We present herein an overview of the proposed Byzantine gathering algorithm. The underlying idea of the algorithm is made of the three following steps:

(1) Each agent a_i starts the rendezvous procedure $\texttt{REN}(a_i.id)$.

(2) When a_i meets another agent a_j with a smaller ID, a_i stops $\texttt{REN}(a_i.id)$ and accompanies a_j.

(3) When a_i executes $\texttt{REN}(a_i.id)$ for $t_{\texttt{REN}}(a_i.id)$ rounds without stopping, a_i and its accompanied agents transition into a terminal state.

If no Byzantine agents exist (non-Byzantine environment), all agents gather at the node where the agent a_{min} with the smallest ID exists, and then transition into the terminal state at the same node at the same time. However, if a Byzantine agent exists (Byzantine environment), that idea fails. Let us consider the case where a_{min} is a Byzantine agent. If a_{min} meets only a part of good agents in Step (1), good agents are divided into two or more groups, and good agents transition into the terminal state at different nodes.

The existing approach by Dieudonné et al. [11], which is tolerant to Byzantine environments, has an agent perform steps (1) and (2). If an agent detects rogue agents behaving as Byzantine at the same node, it does the above, except for the rogue agents. Agents eventually record all IDs of the rogue agents, and this approach guarantees that all good agents meet at the same node. However, it is difficult to terminate agents at the same node at the same time. The authors addressed this problem by using a mechanism that ensures matching IDs for exclusion among good agents at the same node, but the algorithm they proposed takes much time.

We solve this problem in a manner different from the abovementioned approach. To counteract the influence of Byzantine agents, the proposed algorithm has agents form a *reliable group*, which has a special ID, called a *group ID*, and is composed of at least $2f+1$ agents (i.e., this group includes $f+1$ good agents). When an agent meets the reliable group, the agent can trust the group because it understands that at least one good agent belongs to the group. Thus, the proposed algorithm achieves the gathering in the Byzantine environment by modifying the above idea as follows:

(1') After at least one reliable group is created, each reliable group RG with a group ID gid starts $\texttt{REN}(gid)$, and each agent a_i not in the group starts $\texttt{REN}(a_i.id)$.

(2') When a_i or RG meets another reliable group RG' with a smaller group ID, it stops its own rendezvous procedure and accompanies RG'.

(3') When RG executes $\texttt{REN}(gid)$ for $t_{\texttt{REN}}(gid)$ rounds without stopping, RG and its accompanied agents transition into a terminal state.

Consequently, all good agents eventually accompany the reliable group with the smallest group ID and achieve the gathering.

The proposed algorithm creates a reliable group by making agents execute the following steps:

(a) Each agent collects IDs of at least $3f + 1$ good agents and regards them as a *group candidate*.

(b) Agents in the group candidate make a common ID set based on their IDs.

(c) These agents form a reliable group by gathering at the same node based on the common ID set.

A common ID set is used to efficiently form a reliable group. To make the common ID set, a group candidate simulates a parallel consensus algorithm `PCONS`. Once agents in the group candidate make a common ID set, each of them decides on a target ID in the order based on the common ID set and tries to gather one by one at the same node as the agent with the target ID. The algorithm ensures that good agents eventually gather at the same node and create a reliable group.

The gathering algorithm proposed in Part IV employs another strategy for creating a reliable group by collecting IDs. In this algorithm, each good agent searches for one of the agents with the smallest $f + 1$ IDs of the collected IDs to gather at the nodes with the agents. This strategy allows good agents to gather at most $f + 1$ different nodes and requires at least $\Omega(f^2)$ good agents to create a reliable group. By contrast, the proposed algorithm uses the strategy such that $\Omega(f)$ good agents make a common ID set and synchronously search for a target agent one by one to gather at a node with the target agent. Therefore, the algorithm requires $\Omega(f)$ good agents, and the key to reduction is the reliable group creation procedure using the consensus algorithm.

2.2 Algorithms

In this section, we give two algorithms, namely, `MakeReliableGroup` and `ByzantineGathering`. First, we explain Algorithm `MakeReliableGroup` to create a reliable group. Next, we propose Algorithm `ByzantineGathering` that solves the gathering problem with non-simultaneous termination using `MakeReliableGroup`.

2.2.1 Idea of the Algorithm to Create a Reliable Group

Algorithm `MakeReliableGroup` ensures that at least $2f + 1$ agents gather at the same node and form a reliable group. To do this, as mentioned in Section 2.1, `MakeReliableGroup` makes agents in the same group candidate create a common ID set and search for agents with target IDs.

In `MakeReliableGroup`, the agents proceed in five stages: WakeUp, CollectID, MakeCandidate, AgreeID, and MakeGroup. Each stage has one or more *cycles* comprising one or more rounds. The length (the number of rounds) of the first cycle is a given number $T_{ini} > t_{EX}$, and an agent doubles the length every cycle like $2T_{ini}, 4T_{ini}, \ldots$ until the MakeCandidate stage is finished and does not update the length from the AgreeID stage.

In the WakeUp stage, agents wake all dormant agents up. This guarantees that all good agents wake up within t_{EX} rounds. We say two agents start cycles or stages *almost simultaneously* if they start the cycles or stages within t_{EX} rounds.

In the CollectID stage, agents collect IDs, including those of all good agents. Each agent a_i meets the other good agents using the rendezvous procedure when the length of the current cycle is long enough to meet them.

In the MakeCandidate stage, agents create a group candidate. Each agent meets the other agents to confirm that a sufficient number of agents have entered the MakeCandidate stage to transition into the next stage. Agents that transition into the next stage almost simultaneously form a group candidate with each other, guaranteeing that at least $3f + 1$ good agents exist in some group candidate.

In the AgreeID stage, agents collect the IDs of all good agents in the same group candidate. After that, agents in the same group candidate obtain two common ID sets, one from the ID sets collected in the CollectID stage and another from the ID sets collected in the AgreeID stage. Both ID sets are used to efficiently form a reliable group in the next stage. Due to Byzantine agents, agents in the same group candidate may have different ID sets; thus, we use the parallel consensus algorithm `PCONS` to obtain a common ID set. Algorithm

Algorithm 6: `MakeReliableGroup`

$a_i.numRound \leftarrow a_i.numRound + 1$
if $a_i.stage = WakeUp$ **then**
    Execute `WakeUpStage`
else if $a_i.stage = CollectID$ **then**
    // While executing `CollectIDStage`, a_i executes $a_i.lenCycle \leftarrow 2 \cdot a_i.lenCycle$.
    Execute `CollectIDStage`
else if $a_i.stage = MakeCandidate$ **then**
    // While executing `MakeCandidateStage`, a_i executes $a_i.lenCycle \leftarrow 2 \cdot a_i.lenCycle$.
    Execute `MakeCandidateStage`
else if $a_i.stage = AgreeID$ **then**
    // While executing `AgreeIDStage`, a_i executes $a_i.numCycle \leftarrow a_i.numCycle + 1$
    Execute `AgreeIDStage`
else if $a_i.stage = MakeGroup$ **then**
    // While executing `MakeGroupStage`, a_i executes $a_i.numCycle \leftarrow a_i.numCycle + 1$
    Execute `MakeGroupStage`

`PCONS` is for a synchronous message-passing system where, in each phase, each node executes a local computation, sends messages to some nodes, and receives the messages sent. We simulate the behavior of one phase with one cycle. In each simulation cycle, when an agent meets another agent in the same group candidate, they exchange messages in the corresponding phase. The length of a cycle in the AgreeID stage is long enough for any two good agents to meet; hence, good agents can simulate one phase of `PCONS` with one cycle.

In the MakeGroup stage, the agents in a group candidate create a reliable group. Good agents in a group candidate search for target agents one by one in the order based on a common ID set until a sufficient number of agents gather at the same node with a target agent. This algorithm guarantees that at least one group candidate successfully creates a reliable group.

2.2.2 Details of the Algorithm for Creating a Reliable Group

Algorithm 6 shows the behavior of each round of Algorithm `MakeReliableGroup` and executes one of Algorithms 7–12 depending on the current stage. In `MakeReliableGroup`, the procedure `WAIT()` means that an agent stays at the current node for one round. We use function $\text{extendId}(id, bool) = 2 \cdot id + bool$, where id is an agent ID, and $bool$ is a binary integer (0 or 1). In `MakeReliableGroup`, when a_i executes the rendezvous procedure with ID id_i, a_i uses $\text{extendId}(id_i, 0)$

Table 4. Variables of agent a_i (Part 1).

Variable	Initial value	Explanation
lenCycle	$T_{ini}(> t_{EX})$	Length of the current cycle
stage	*WakeUp*	Current stage of a_i. This variable takes one of the following values: *WakeUp*, *CollectID*, *MakeCandidate*, *AgreeID*, and *MakeGroup*
numRound	0	The number of rounds from the beginning of the WakeUp stage or the beginning of the current cycle
ready	*False*	*True* if and only if a_i has met Condition (1) or (2) of Algorithm 9
R	$\varnothing$	A set of IDs of agents such that a_i knows they satisfy *ready* = *True*
S_p	$\{a_i.id\}$	A set of agent IDs that a_i has collected in the CollectID stage
endMakeCandidate	*False*	*True* if and only if a_i can transition into the AgreeID stage
P_p	$\varnothing$	A set of IDs of agents such that a_i knows they belong to the same group candidate as a_i

as the input of the rendezvous procedure. Tables 4 and 5 summarizes variables in `MakeReliableGroup`. Agent a_i doubles $a_i.lenCycle$ at the end of each cycle if $a_i.stage \in \{CollectID, MakeCandidate\}$.

We focus on the progress of stages and cycles of good agents. The overall flow of `MakeReliableGroup` is shown in Fig. 3. In this figure, symbols W, C, M, and A represent the cycles of the WakeUp stage, CollectID stage, MakeCandidate stage, and AgreeID stage, respectively. Note that the scale is different for the upper and lower figures. An agent executes the WakeUp, CollectID, MakeCandidate, AgreeID, and MakeGroup stages in this order. Every good agent operates at the WakeUp stage for t_{EX} rounds. The behavior of the WakeUp stage guarantees that all good agents start the CollectID stage almost simultaneously. In the CollectID and MakeCandidate stages, all good agents double their length in the last round of each cycle. Hence, all good agents in the CollectID or MakeCandidate stage start their cycles almost simultaneously and have the same cycle length. The following observation formally shows this fact. We denote the γ-th cycle of an agent a_i by c_i^γ. The length of cycle c_i^γ is the value of $a_i.lenCycle$ at the beginning of cycle c_i^γ and is represented as $|c_i^\gamma|$. Variables $c_i^\gamma[j]$ and $c_i^\gamma[last]$ represent the j-th round and the last round of cycle c_i^γ, respectively.

Table 5. Variables of agent a_i (Part 2).

Variable	Initial value	Explanation
numCycle	0	The number of cycles from the beginning of the AgreeID stage
S_c	$\varnothing$	An output of `PCONS`(S_p)
P_c	$\varnothing$	An output of `PCONS`(P_p) used as a common ID set, with elements ordered in an increasing order
D	$\varnothing$	A set of a combination $(id, numRR)$, where id is an ID of an agent that has met all conditions of Function *satisfyCRG* at the current node, and *numRR* is the number of rounds for the agent with id to finish its current cycle
numRemainRound	∞	The number of rounds remaining before all good agents in a reliable group finish a cycle at the same time
guidepostId	∞	The ID for agents in the same group candidate to move together in the MakeGroup stage
BL	$\varnothing$	A set of IDs of agents that behave inappropriately during the reliable group formation
gid	∞	The group ID of the reliable group to which a_i belongs

Observation **2.** *Let a_i and a_j be two different good agents in the* CollectID *or* MakeCandidate *stage. Agents a_i and a_j start their cycles c_i^γ and c_j^γ almost simultaneously, and $|c_i^\gamma| = |c_j^\gamma|$ holds.*

In the AgreeID and MakeGroup stages, good agents do not update the length of their cycles. Good agents may transition into the AgreeID stage at different cycles. Therefore, each good agent may update the length of its cycle by a different number of times; thus, good agents in the AgreeID or the MakeGroup stage may have different cycle lengths.

Algorithm `MakeReliableGroup` does not guarantee that two different agents a_i and a_j start their cycle at the same time. Thus, even in the case where a_i and a_j execute cycles c_i^γ and c_j^γ, respectively, they may execute different cycles during the first and last t_{EX} rounds of their cycles. Therefore, in `MakeReliableGroup`, a_i cooperates with a_j only during a cycle excluding the first and last t_{EX} rounds of cycle c_i^γ. We call this period a *core period* of a cycle.

WakeUp Stage Algorithm 7 is the pseudo-code of the WakeUp stage. This stage aims to wake all dormant agents up. To do this, agent a_i explores the network using `EX`. Agent a_i then updates its variables at the beginning of the last

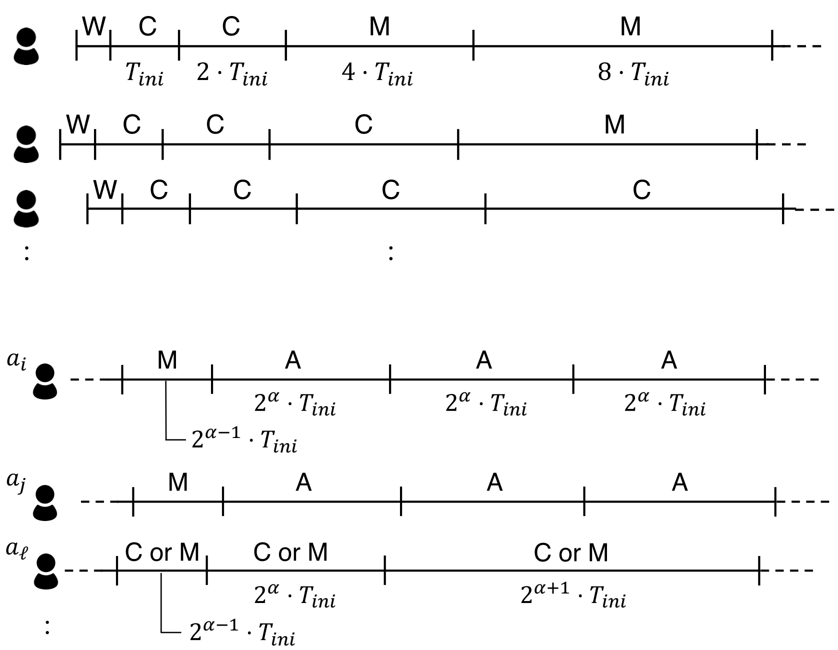

Figure 3. Stage flow of Algorithm `MakeReliableGroup` (Upper: Starting this algorithm, Lower: Starting AgreeID stage).

round of exploring. Agent a_i visits all nodes by the end of this stage; hence, at least all dormant good agents start the algorithm. This stage guarantees that at least all good agents start the CollectID stage almost simultaneously.

CollectID Stage Algorithm 8 is the pseudo-code of the CollectID stage. This stage aims to collect the IDs of all good agents. Recall that A_i is a set of agents, including a_i, that stay at the current node of a_i at the beginning of a round.

Agent a_i collects IDs of agents with $ready = True$ at the beginning of a round during a core period of a cycle. Agent a_i uses variable R to record these IDs. We will explain the details of variables *ready* and R in the description of the MakeCandidate stage.

If $a_i.lenCycle < 6 \cdot (t_{\texttt{REN}}(\text{extendId}(a_i.id, 0)) + 1)$ holds, a_i stays at the current node for the current cycle. Otherwise, if $a_i.lenCycle \geq 6 \cdot (t_{\texttt{REN}}(\text{extendId}(a_i.id, 0)) + 1)$ holds, a_i collects IDs, including those of all good agents using `REN`(extendId($a_i.id, 0$)) and stores the collected IDs in $a_i.S_p$. In

Algorithm 7: `WakeUpStage`

if $a_i.numRound < t_{EX}$ **then**
  Execute $\texttt{EX}(a_i.numRound)$
else
  $a_i.numRound \leftarrow 0$
  $a_i.stage \leftarrow CollectID$
  Execute $\texttt{EX}(t_{EX})$

Algorithm 8: `CollectIDStage`

if $t_{EX} \le a_i.numRound \le a_i.lenCycle - t_{EX}$ **then**
  $a_i.R \leftarrow a_i.R \cup \{a_j.id \mid a_j \in A_i \wedge a_j.ready = True\}$
if $a_i.lenCycle < 6 \cdot (t_{REN}(\text{extendId}(a_i.id, 0)) + 1)$ **then**
  if $a_i.numRound = a_i.lenCycle$ **then**
    $a_i.numRound \leftarrow 0$
    $a_i.lenCycle \leftarrow 2 \cdot a_i.lenCycle$
  Execute $\texttt{WAIT}()$
else /* $a_i.lenCycle \ge 6 \cdot (t_{REN}(\text{extendId}(a_i.id, 0)) + 1)$ */
  $a_i.S_p \leftarrow a_i.S_p \cup \{a_j.id \mid a_j \in A_i\}$
  if $a_i.numRound < a_i.lenCycle$ **then**
    Execute $\texttt{REN}(\text{extendId}(a_i.id, 0))(a_i.numRound)$
  else
    $a_i.numRound \leftarrow 0$
    $a_i.lenCycle \leftarrow 2 \cdot a_i.lenCycle$
    $a_i.stage \leftarrow MakeCandidate$
    Execute $\texttt{WAIT}()$

both cases, a_i updates its variables at the beginning of the last round of a cycle. If a_i has started $\texttt{REN}(\text{extendId}(a_i.id, 0))$ at the beginning of the current cycle, a_i executes $a_i.stage \leftarrow MakeCandidate$ and waits for one round in the last round of the current cycle.

MakeCandidate Stage Algorithm 9 is the pseudo-code of the MakeCandidate stage. This stage aims to create a group candidate comprising at least $3f + 1$ good agents. To describe this stage clearly, we define the group candidate as follows:

Definition **2** (Group candidate)**.** *A set GC of good agents is a group candidate if and only if GC is a maximal set of good agents that start the* AgreeID *stage almost simultaneously.*

Algorithm 9: `MakeCandidateStage`

Function inferIdsAfterMC($a_i.S_p, a_i.lenCycle$) = $\{id \in a_i.S_p \mid a_i.lenCycle \geq 12 \cdot (t_{REN}(\text{extendId}(id, 0)) + 1)\}$: Assuming that a_i executes a cycle c_i^γ as the current cycle, this returns IDs of agents, from $a_i.S_p$, each a_j of which starts the MakeCandidate stage by a cycle c_j^γ.

if $t_{EX} \leq a_i.numRound \leq a_i.lenCycle - t_{EX}$ **then**
    $a_i.R \leftarrow a_i.R \cup \{a_j.id \mid a_j \in A_i \wedge a_j.ready = True\}$
if $a_i.numRound = 1 \wedge$ (*either (1) or (2) holds*) $\wedge\ a_i.ready = False$ **then**
    // (1) $|\text{inferIdsAfterMC}(a_i.S_p, a_i.lenCycle)| \geq (7/8)|a_i.S_p|$
    // (2) $|a_i.R| \geq (1/2)|a_i.S_p|$
    $a_i.ready \leftarrow True$
    $a_i.R \leftarrow a_i.R \cup \{a_i.id\}$
if $a_i.numRound = 1 \wedge |a_i.R| \geq (3/4)|a_i.S_p|$ **then**
    $a_i.endMakeCandidate \leftarrow True$
if $a_i.numRound < a_i.lenCycle$ **then**
    Execute `REN`(extendId($a_i.id, 0$))($a_i.numRound$)
else
    $a_i.numRound \leftarrow 0$
    $a_i.lenCycle \leftarrow 2 \cdot a_i.lenCycle$
    if $a_i.endMakeCandidate = True$ **then**
        $a_i.stage \leftarrow AgreeID$
    Execute `WAIT`()

By Observation 2, for another good agent a_j in the MakeCandidate stage, a_i and a_j start their cycles of the MakeCandidate stage almost simultaneously. Therefore, if a_i and a_j start the AgreeID stage in cycles c_i^γ and c_j^γ for a positive integer γ, respectively, they satisfy the group candidate requirement. Consequently, if at least $3f + 1$ good agents start the AgreeID stage almost simultaneously, `MakeReliableGroup` achieves the purpose of this stage.

We explain the detail of the MakeCandidate stage hereinafter. Agent a_i executes `REN`(extendId($a_i.id, 0$)) during a cycle, except for the last round to meet all good agents. Agent a_i then waits for one round to update its variables in the last round of a cycle. In parallel with the above executions, a_i executes the following:

As with the CollectID stage, a_i collects IDs of agents with $ready = True$ at the beginning of a round during a core period of a cycle.

At the beginning of the first round of a cycle, if a_i satisfies either of the two following conditions, a_i stores *True* in $a_i.ready$ to claim that a sufficient number of agents satisfy a condition to finish the MakeCandidate stage.

Algorithm 10: `AgreeIDStage`

Function detectGC($A_i, a_i.lenCycle$) = $\{a_j.id \mid a_j \in A_i \wedge a_j.lenCycle = a_i.lenCycle \wedge a_j.stage = AgreeID\}$: This returns IDs of agents in the same group candidate as a_i at the current node.

if $a_i.numCycle = 0$ **then**
  $a_i.P_p \leftarrow a_i.P_p \cup$ detectGC($A_i, a_i.lenCycle$)
else
  Execute `PCONS`($a_i.S_p$)($a_i.numCycle$)
  Execute `PCONS`($a_i.P_p$)($a_i.numCycle$)
if $a_i.numRound < a_i.lenCycle$ **then**
  Execute `REN`(extendId($a_i.id, 0$))($a_i.numRound$)
else
  $a_i.numRound \leftarrow 0$
  $a_i.numCycle \leftarrow a_i.numCycle + 1$
  if *`PCONS`($a_i.S_p$) and `PCONS`($a_i.P_p$) are finished* **then**
    $a_i.S_c \leftarrow$ the output of `PCONS`($a_i.S_p$)
    $a_i.P_c \leftarrow$ the output of `PCONS`($a_i.P_p$)
    $a_i.stage \leftarrow MakeGroup$
  Execute `WAIT`()

Ready-Condition (1) Variable $a_i.S_p$ contains at least $(7/8)|a_i.S_p|$ IDs of agents that have started the MakeCandidate stage.

Ready-Condition (2) Agent a_i witnessed at least $(1/2)|a_i.S_p|$ agents with $ready = True$ from the beginning of `MakeReliableGroup`.

To verify Ready-Condition (1), a_i checks whether or not the result of a function inferIdsAfterMC($a_i.S_p, a_i.lenCycle$) contains at least $(7/8)|a_i.S_p|$ IDs. To verify Ready-Condition (2), a_i checks whether or not $a_i.R$ includes at least $(1/2)|a_i.S_p|$ IDs. After checking Ready-Conditions (1) and (2), if $a_i.R$ contains at least $(3/4)|a_i.S_p|$ IDs, a_i stores *True* in $a_i.endMakeCandidate$. It means that a_i starts the AgreeID stage from the next cycle.

Agent a_i updates its variables at the beginning of the last round of a cycle. If $a_i.endMakeCandidate = True$ holds, a_i stores AgreeID in $a_i.stage$.

By the behavior of this stage, we have the following observation:

Observation **3.** *Two different good agents in a group candidate always start their cycles almost simultaneously.*

AGREEID Stage Algorithm 10 is the pseudo-code of the AGREEID stage. This stage aims to obtain two common ID sets among good agents in a group candidate by using the consensus algorithm `PCONS`. One contains IDs of all good agents in the same group candidate, but not IDs of the other good agents, while the other contains IDs of all good agents. Agents in the group candidate use these common ID sets to form a reliable group in the MAKEGROUP stage. The former common ID set is used to efficiently form a reliable group formation. To make the former common ID set, agents collect IDs of good agents in the same group candidate and make a consensus on the collected IDs.

We will explain the details of the AGREEID stage hereinafter. Agent a_i executes `REN`(extendId($a_i.id, 0$)) during a cycle, except for the last round to meet all good agents. Agent a_i then waits for one round to update its variables in the last round of a cycle. In parallel with the above executions, a_i executes the following:

If a_i executes the first cycle of this stage, a_i collects agent IDs in the same group candidate, say GC, and stores these IDs in P_p. To collect these IDs, a_i uses a function detectGC($A_i, a_i.lenCycle$).

If a_i executes the second or later cycle of this stage, a_i makes a consensus on two ID sets $a_i.P_p$ and $a_i.S_p$ using `PCONS`($a_i.P_p$) and `PCONS`($a_i.S_p$) to obtain two common ID sets with good agents in GC. As mentioned in Section 2.2.1, agents simulate one phase of the message-passing model by executing `REN` for one cycle. In Algorithm 10, a_i executes `PCONS`(S)(p) during a cycle c_i^γ, except for the last round of this cycle. More concretely, a_i makes a message *msg* of a phase p in round $c_i^\gamma[1]$. Agent a_i then sends *msg* for other agents in GC between rounds $c_i^\gamma[2]$ and $c_i^\gamma[last-1]$. When a_i receives the messages sent by an agent a_ℓ at the current node between rounds $c_i^\gamma[2]$ and $c_i^\gamma[last-1]$, a_i records the messages and $a_\ell.id$ only if $a_i.lenCycle = a_\ell.lenCycle$ and $a_i.numCycle = a_\ell.numCycle$ hold.

At the beginning of the last round of a cycle, a_i updates its variables and confirms the status of `PCONS`($a_i.P_p$) and `PCONS`($a_i.S_p$). If both the consensus instances have finished, a_i stores the outputs in $a_i.S_c$ and $a_i.P_c$ and executes $a_i.stage \leftarrow MakeGroup$.

MAKEGROUP Stage Algorithm 12 is the pseudo-code of the MAKEGROUP stage. In this stage, the agents in a group candidate form a reliable group at the

Algorithm 11: Functions of the MakeGroup stage of agent a_i.

Function target($a_i.P_c, a_i.numCycle$) = $a_i.P_c[a_i.numCycle \mod |a_i.P_c|]$: This returns the ($a_i.numCycle \mod |a_i.P_c|$)-th smallest element in $a_i.P_c$.

Function satisfyCRG($a_j, a_i.lenCycle, a_i.S_c$) = $(|a_j.S_c| \geq (7/8)|a_j.S_p| \wedge a_j.lenCycle = a_i.lenCycle \wedge a_j.S_c = a_i.S_c \wedge a_j.stage = MakeGroup \wedge a_j.numRound \leq (1/2) \cdot a_i.lenCycle \wedge a_j.numRemainRound = \infty)$: This returns whether a_j is in a sufficient state to become a member of the same reliable group as a_i.

Function median($\{x_2 \mid (x_1, x_2) \in a_i.D\}$): This returns the median of $\{x_2 \mid (x_1, x_2) \in a_i.D\}$ if $|\{x_2 \mid (x_1, x_2) \in a_i.D\}|$ is odd, and otherwise returns the rounded-up arithmetic mean of two middle values.

Function detectByzantine($A_i, \{x_1 \mid (x_1, x_2) \in a_i.D\}$) = $\{a_j.id \in \{x_1 \mid (x_1, x_2) \in a_i.D\} \mid a_j \notin A_i \vee a_j.numRemainRound = \infty\}$: This returns IDs of agents, from $\{x_1 \mid (x_1, x_2) \in a_i.D\}$, each a_j of which does not exist at the current node or initializes its variables.

same node in round r_{fg}, which exists within t_{EX} rounds right after the first good agent in the reliable group finishes a cycle of the MakeGroup stage. The agents also do not create multiple reliable groups with the same group ID. Algorithm 11 summarizes functions in this stage. An agent a_i has variable *gid* to keep the group ID of the reliable group to which a_i belongs.

In Algorithm `ByzantineGathering`, agents do not know f. Thus, when an agent a_i determines whether or not a reliable group exists at the current node, a_i uses $(1/7)|a_i.S_p|$ instead of f for its decision. However, $|a_i.S_p|$ may differ from the other good agents because every good agent has possibly met a different number of Byzantine agents in the CollectID stage. To recognize a reliable group, even if a good agent has any S_p, we define a reliable group as follows:

Definition **3** (Reliable group)**.** *A set RG of agents is a reliable group if and only if it is a maximal set such that it contains at least $k/7$ good agents, and every pair of distinct good agents $a_i, a_j \in RG$ has the same group ID ($a_i.gid = a_j.gid$).*

This stage ensures that at least $k/7$ good agents with the same cycle length simultaneously form a reliable group. All the good agents of the reliable group then have the same group ID. Hence, if the good agents of the reliable group start rendezvous procedures with their group ID for a duration corresponding to the cycle length, starting from the next round after the creation of the reliable group, they are always located at the same node during this period.

First, we give the behavior of this stage of an agent a_i in a high-level way. Let GC be a group candidate of a_i. As long as a_i has not gathered at a single node

Algorithm 12: `MakeGroupStage`

if $a_i.numRemainRound = \infty \land a_i.numRound \leq (1/2) \cdot a_i.lenCycle$ **then**
    if $\exists a_j \in A_i[a_j.id = \text{target}(a_i.P_c, a_i.numCycle)]$ **then**
        $a_i.D \leftarrow \{(a_j.id, a_j.lenCycle - a_j.numRound) \mid a_j \in A_i \land \text{satisfyCRG}(a_j, a_i.lenCycle, a_i.S_c) = True\}$
        if $|a_i.S_c| \geq (7/8)|a_i.S_p| \land |a_i.D| \geq (3/8)|a_i.S_c| \land \text{median}(\{x_2 \mid (x_1, x_2) \in a_i.D\}) \geq (1/2) \cdot a_i.lenCycle$ **then**
            $a_i.numRemainRound \leftarrow \text{median}(\{x_2 \mid (x_1, x_2) \in a_i.D\})$
            $a_i.guidepostId \leftarrow \min(\{x_1 \mid (x_1, x_2) \in a_i.D\})$
        Execute `WAIT`()
    else
        Execute `REN`(extendId($a_i.id, 0$))($a_i.numRound$)
else
    if $a_i.numRemainRound \neq \infty$ **then**
        $a_i.numRemainRound \leftarrow a_i.numRemainRound - 1$
        $a_i.BL \leftarrow a_i.BL \cup \text{detectByzantine}(A_i, \{x_1 \mid (x_1, x_2) \in a_i.D\})$
        if $a_i.numRemainRound > 0$ **then**
            Execute `REN`(extendId($a_i.guidepostId, 0$))($a_i.numRound$)
        else
            $a_i.numRound \leftarrow 0$
            $a_i.gid \leftarrow \min(\{x_1 \mid (x_1, x_2) \in a_i.D\} \setminus a_i.BL)$
            Execute `WAIT`()
    else
        if $a_i.numRound < a_i.lenCycle$ **then**
            Execute `REN`(extendId($a_i.id, 0$))($a_i.numRound$)
        else
            $a_i.numRound \leftarrow 0$
            $a_i.numCycle \leftarrow a_i.numCycle + 1$
            Execute `WAIT`()

with enough agents to form a reliable group in the first half of a cycle, called the first-subcycle, a_i behaves in the first-subcycle for agents in GC to gather at a single node and executes `REN`(extendId($a_i.id, 0$)) in the second half of a cycle, called the second-subcycle, to meet the other good agents. Henceforth, we simply call enough agents to form a reliable group, called "sufficient agents." Once a_i gathers with sufficient agents at a single node in the first-subcycle, a_i acts with the gathered agents in the rest of this cycle to meet the other good agents and simultaneously form a reliable group. The behavior of meeting the other good agents is necessary in guaranteeing that an agent meets all good agents in the CollectID stage and all reliable groups.

To gather with sufficient agents in the first-subcycle, a_i decides a target ID by using variables $a_i.P_c$ and $a_i.numCycle$. If $a_i.id$ is not the target ID, a_i searches for the agent with the target ID, say a_{target}, using $\texttt{REN}(\text{extendId}(a_i.id, 0))$; otherwise, a_i stays at the current node. We denote the node with a_{target} as v_{target}. If a_i does not gather with sufficient agents at v_{target} by the end of the first-subcycle, a_i abstains from the group creation and executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$ in this cycle to meet the other good agents. It then decides a new target ID and executes the group creation using the new target ID in the next cycle; otherwise, along with the gathered agents, a_i determines the round r_{fg} to simultaneously form a reliable group with the gathered agents. To decide round r_{fg}, a_i shares with the gathered agents the remaining round to finish the current cycle and computes the median of the remaining rounds. Here, for some gathered agents, round r_{fg} may not be the last round of their current cycle. Thus, only in the cycle that agents decide round r_{fg} do the gathered agents regard round r_{fg} as the last round of their current cycle.

After deciding on round r_{fg}, a_i explores the network with the gathered agents until round r_{fg} to meet the other good agents. To do this, a_i executes a rendezvous procedure with the smallest ID among the gathered agents. We represent this ID as id_{min}. While executing the rendezvous procedure with id_{min}, the gathered agents monitor each other to notice that a part of them has left and behaved improperly. When reaching round r_{fg}, a_i becomes a member of a reliable group along with the agents that have behaved correctly. It then determines the smallest ID among IDs of members of the reliable group as a group ID. The group ID is selected among IDs of agents that start the rendezvous procedure with id_{min} together; thus, this behavior guarantees that the group ID is unique.

We will now explain the details of this stage of agent a_i. First, we describe the behavior of the first-subcycle for a_i to gather with sufficient agents. At the beginning of a first-subcycle round, a_i checks whether or not a_i satisfies either of the two following conditions at the current node.

Target-Condition (1) Agent a_i has a target ID.

Target-Condition (2) Agent a_i meets a_{target} at the current node.

Agent a_i adopts $\text{target}(a_i.P_c, a_i.numCycle)$ as a target ID. Recall that a_i counts the number of cycles from the start of the AgreeID stage by $a_i.numCycle$. By Observation 3, all good agents in GC always start their cycles almost simultaneously; thus, all good agents in GC have experienced the same number of cycles and have the same $numCycle$ during a core period of a cycle. Furthermore, if GC contains at least $3f+1$ good agents, all good agents in GC have the same P_c. Hence, target$(a_i.P_c, a_i.numCycle)$ guarantees that all good agents in GC decide the same target ID.

If a_i satisfies either Target-Condition (1) or (2), a_i checks whether there are sufficient agents at the current node, and then a_i stays at the current node as long as a_i satisfies either Target-Condition (1) or (2); otherwise, a_i executes `REN`(extendId$(a_i.id, 0)$) until a_i meets a_{target}. To check whether there are sufficient agents, a_i verifies whether each agent a_j at the current node is in a sufficient state to become a member of the same reliable group as a_i in the current cycle by checking the variables of a_j. To check the state of a_j, a_i uses satisfyCRG$(a_j, a_i.lenCycle, a_i.S_c)$. If satisfyCRG$(a_j, a_i.lenCycle, a_i.S_c) = True$ holds, a_i determines that a_j is in a sufficient state to become a member of the same reliable group as a_i and stores $(a_j.id, a_j.lenCycle - a_j.numRound)$ in $a_i.D$. Note that element $a_j.lenCycle - a_j.numRound$ is the remaining round to finish the current cycle of a_j.

After updating $a_i.D$, a_i checks whether a_i satisfies the three following conditions:

NumRR-Condition (1) Variable $a_i.S_c$ contains at least $(7/8)|a_i.S_p|$ IDs.

NumRR-Condition (2) Variable $a_i.D$ contains at least $(3/8)|a_i.S_c|$ tuples.

NumRR-Condition (3) The median of $\{x_2 \mid (x_1, x_2) \in a_i.D\}$ is at least $(1/2) \cdot a_i.lenCycle$.

If a_i does not satisfy any of NumRR-Conditions (1)–(3) by the last round of the first-subcycle, a_i executes `REN`(extendId$(a_i.id, 0)$) during the second-subcycle, except for the last round to meet all good agents in the CollectID stage and all reliable groups. Agent a_i then waits for one round in the last round of the second-subcycle. In this case, a_i is not involved in a reliable group formation in the current cycle.

If a_i satisfies all of NumRR-Conditions (1)–(3) by the last round of the first-subcycle, a_i decides that there are sufficient agents and determines round r_{fg} along with the agents in $\{x_1 \mid (x_1, x_2) \in a_i.D\}$. To determine round r_{fg}, a_i stores the median of $\{x_2 \mid (x_1, x_2) \in a_i.D\}$ in $a_i.numRemainRound$. At the same time, a_i stores the smallest ID of $\{x_1 \mid (x_1, x_2) \in a_i.D\}$ in $a_i.guidepostId$ to move with the agents in $\{x_1 \mid (x_1, x_2) \in a_i.D\}$. From the next round when a_i updates $a_i.numRemainRound$ and $a_i.guidepostId$, a_i decreases the value of $a_i.numRemainRound$ by one in each subsequent round. To meet all good agents in the CollectID stage and all reliable groups, a_i executes `REN`(extendId($a_i.guidepostId, 0$)) as long as $a_i.numRemainRound = 0$ does not hold. In parallel with these executions, a_i monitors agents in $\{x_1 \mid (x_1, x_2) \in a_i.D\}$ to expose the Byzantine agents in $\{x_1 \mid (x_1, x_2) \in a_i.D\}$. To detect the Byzantine agents, a_i uses a function detectByzantine($A_i, \{x_1 \mid (x_1, x_2) \in a_i.D\}$) that detects the absence of a_j or the initialization of $a_j.numRemainRound$ for an agent a_j in $\{x_1 \mid (x_1, x_2) \in a_i.D\}$. After Byzantine agents store an ID in *guidepostId* with some good agents, say GA_{early}, they must take either of the following actions to store a different ID in *guidepostId* with good agents, say GA_{later}, that will update *guidepostId* later: (1) to meet GA_{later}, the Byzantine agents leave GA_{early}, or (2) to store the different ID in *guidepostId* with GA_{later}, the Byzantine agents initialize *numRemainRound* when they are with GA_{early}. However, these actions are clearly a dishonest behavior. Thus, a_i exposes Byzantine agents in $\{x_1 \mid (x_1, x_2) \in a_i.D\}$ by detecting these actions for agents in $\{x_1 \mid (x_1, x_2) \in a_i.D\}$. Agent a_i stores detectByzantine($A_i, \{x_1 \mid (x_1, x_2) \in a_i.D\}, a_i.lenCycle$) in $a_i.BL$. If $a_i.numRemainRound = 0$ holds, a_i updates $a_i.numRound$, stores the smallest ID among IDs in $\{x_1 \mid (x_1, x_2) \in a_i.D\} \setminus a_i.BL$ in $a_i.gid$, and waits for one round at the current node.

2.2.3 Idea of the Algorithm to Gather

In Algorithm `ByzantineGathering`, all agents execute `MakeReliableGroup`, and then some agents eventually form a reliable group. Subsequently, good agents in the reliable group collect all good agents using `REN` with its group ID. To do this, if an agent not in the reliable group meets the reliable group, it forces `MakeReliableGroup` to terminate and accompanies the reliable group. If agents

Algorithm 13: `ByzantineGathering`(N) for agent a_i

```
if a_i.stage ∈ {WakeUp, CollectID} then
    Execute MakeReliableGroup
else                        /* a_i.stage ∈ {MakeCandidate, AgreeID, MakeGroup} */
    a_i.S_gid ← {x | ∃A_rg ⊂ A_i[|A_rg| ≥ (1/7)|a_i.S_p| ∧ ∀a_j ∈ A_rg : a_j.gid = x ∧ a_j.gid ≠ ∅]}
    if a_i.S_gid ≠ ∅ then
        a_i.minGID ← min(a_i.S_gid)
    if a_i.S_gid ≠ ∅ ∧ a_i.gid > a_i.minGID then
        a_i.S_rg ← {id | ∃a_j ∈ A_i[a_j.gid = a_i.minGID ∧ a_j.id = id]}
        Execute FOLLOW(a_i.S_rg)
    else if a_i.gid ≠ ∞ then
        a_i.numRound ← a_i.numRound + 1
        if a_i.numRound = a_i.lenCycle then
            Execute TERMINATE()
        Execute REN(extendId(a_i.gid, 1))(a_i.numRound)
    else
        Execute MakeReliableGroup
```

in the reliable group meet another reliable group, they accompany the reliable group with the smaller group ID. This algorithm guarantees that all good agents gather at the node, including the reliable group with the smallest group ID, and transition into a terminal state.

2.2.4 Details of the Algorithm to Gather

Algorithm 13 shows the behavior of each round of Algorithm `ByzantineGathering`. This algorithm aims to make all good agents transition into a terminal state using a reliable group. Agent a_i refers to the variables in `MakeReliableGroup`. In Algorithm 13, we introduce two procedures, that is, `TERMINATE()` and `FOLLOW`(S), for an ID set S. Procedure `TERMINATE()` means that an agent transitions into a terminal state. Procedure `FOLLOW`(S) means that an agent a_i executes the two following actions: (1) When the majority of agents in S move to some node, a_i also moves to the node. (2) When the majority of agents in S execute `TERMINATE()` or have entered a terminal state, a_i executes `TERMINATE()`.

Note that an agent employs a different ID when executing the rendezvous procedure as a member of a reliable group. More concretely, an agent uses an ID extended by 1 for the rendezvous procedure. Recall that an agent uses an

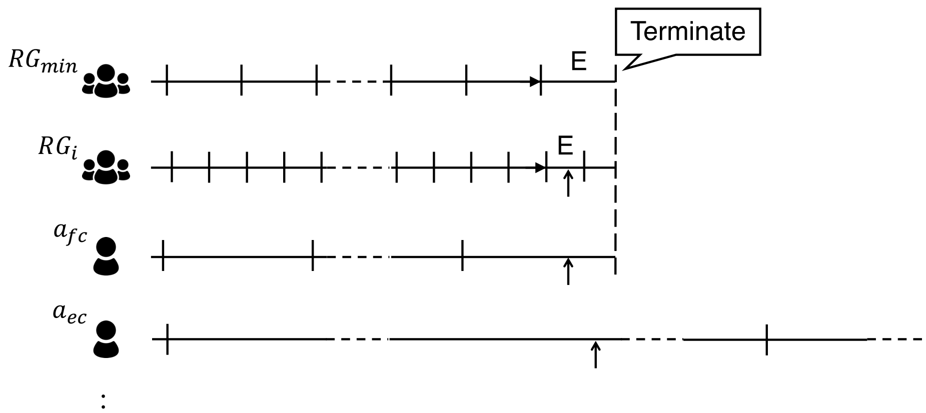

Figure 4. Gathering flow of Algorithm `ByzantineGathering`.

ID extended by 0 for the rendezvous procedure in `MakeReliableGroup`. Thus, agents execute the rendezvous procedure with different extended IDs.

We show the gathering flow with reliable groups in Fig. 4. In this figure, symbol E denotes the execution of a rendezvous procedure with a group ID. The right arrow indicates that agents in a reliable group execute `MakeReliableGroup` until the point specified by the arrowhead. The up arrow signifies that agents in a reliable group and those not in any reliable group meet a reliable group with a smaller group ID than their group ID and a reliable group, respectively. A good agent a_j in any reliable group collects the other good agents using `REN`(extendId($a_j.gid$, 1)). The execution period is determined by $a_j.lenCycle$, that is, the length of a cycle after starting the AgreeID stage. The reliable group RG_{min} has the smallest group ID among reliable groups; thus, RG_{min} does not meet a reliable group with a smaller group ID. Group RG_{min} executes collecting good agents to the end and transitions into a terminal state. By contrast, when another reliable group RG_i meets a reliable group with a smaller group ID, RG_i stops collecting good agents and accompanies it. Group RG_i then eventually meets and accompanies RG_{min} and concurrently transitions into a terminal state when RG_{min} transitions into a terminal state. Consider a good agent a_{fc} that does not belong to any reliable group and finishes the CollectID stage and a good agent a_{ec} that does not belong to any reliable group and is in the CollectID stage. Similar to RG_i, a_{fc} stops executing `MakeReliableGroup` and accompanies a reliable group with a smaller group ID when a_{fc} meets it. Agent a_{fc} then concurrently transitions into a terminal state when RG_{min} transitions into a terminal state. In contrast, a_{ec} continues to execute `MakeReliableGroup`, even if it meets a reliable group. After starting the MakeCandidate stage, a_{ec} then visits the node with RG_{min} using `REN` and transitions into a terminal state.

We explain `ByzantineGathering` hereinafter. Agent a_i executes `MakeReliableGroup` in conjunction with `ByzantineGathering`. If a_i is in the WakeUp or CollectID stage, a_i continues `MakeReliableGroup`. If a_i is in the MakeCandidate, AgreeID, or MakeGroup stage, a_i determines whether a reliable group exists at the current node at the beginning of the current round. If a_i witnesses $(1/7)|a_i.S_p|$ agents with the same gid ($\neq \infty$), a_i recognizes the reliable group and stores their gid in $a_i.S_{gid}$. After that, a_i stores the smallest group ID among $a_i.S_{gid}$ in $a_i.minGID$.

When the node with a_i contains a reliable group with a group ID smaller than $a_i.gid$, a_i follows the group by using `FOLLOW`$(a_i.S_{rg})$. More concretely, a_i finds agents whose gid are equal to $a_i.minGID$ and stores the IDs of these agents in $a_i.S_{rg}$. Agent a_i then follows the action of the majority of agents in $a_i.S_{rg}$ using `FOLLOW`$(a_i.S_{rg})$.

When a_i belongs to a reliable group with the smallest group ID among $a_i.S_{gid}$, a_i executes `REN`(extendId$(a_i.gid, 1)$) for $a_i.lenCycle$ rounds to meet all good agents. If a_i meets a reliable group with a group ID smaller than $a_i.gid$, it follows the reliable group. Note that agents in a reliable group use extended IDs different from those not in a reliable group for the rendezvous procedure; hence, they can meet during the execution of the rendezvous procedure. If a_i does not meet a reliable group with a group ID smaller than $a_i.gid$ and finishes the execution of `REN`(extendId$(a_i.gid, 1)$), a_i transitions into a terminal state using `TERMINATE()`.

If a_i does not satisfy the abovementioned conditions, that is, a reliable group does not exist at the current node, a_i executes `MakeReliableGroup` for one round.

2.3 Correctness and Complexity Analysis

In this subsection, we prove the correctness and the complexity of the proposed algorithm. First, we assume that no reliable group exists in the network and prove that at least one reliable group is created in Section 2.3.1. Next, we prove that all good agents gather and transition into a terminal state at the same node in Section 2.3.2.

2.3.1 Creation of Reliable Groups

In this section, no reliable groups exist in the network, therefore, all agents execute `MakeReliableGroup`.

We will first focus on the relationship of cycles among agents. Let a_i and a_j be two different good agents. Assume that a_i is in the CollectID or MakeCandidate stage in cycle c_i^γ. By Observation 2, if a_j is in the CollectID or MakeCandidate stage in cycle c_j^γ, a_i and a_j start cycles c_i^γ and c_j^γ almost simultaneously, and $|c_i^\gamma| = |c_j^\gamma|$ holds. By contrast, the length of a cycle in the AgreeID or MakeGroup stage depends on when the agent starts the AgreeID stage. When a_i and a_j start the AgreeID stage almost simultaneously in cycles c_i^η and c_j^η, they do not update $a_i.lenCycle$ and $a_j.lenCycle$ later; hence, $|c_i^\epsilon| = |c_j^\epsilon|$ holds for $\varepsilon \geq \eta$. By contrast, when a_i and a_j start the AgreeID stage in cycles c_i^η and $c_j^{\eta'}$ ($\eta < \eta'$), respectively, only a_j updates $a_j.lenCycle$ after c_j^η. Hence, $|c_i^\varepsilon| \neq |c_j^{\varepsilon'}|$ holds for $\varepsilon \geq \eta$ and $\varepsilon' \geq \eta'$. The following observation formally shows this fact:

Observation **4.** *Let a_i and a_j be good agents. Let c_i^γ and $c_j^{\gamma'}$ be a cycle of the* AgreeID *or* MakeGroup *stage of a_i and a_j, respectively. If and only if a_i and a_j start the* AgreeID *stage almost simultaneously does $|c_i^\gamma| = |c_j^{\gamma'}|$ hold.*

We also focus on the case where a_j starts the AgreeID stage not more than t_{EX} rounds later than a_i. In this case, a_i starts the AgreeID stage in a cycle c_i^η, and a_j starts that stage in a cycle c_j^ζ for $\eta < \zeta$. Agent a_i does not update $a_i.lenCycle$ from cycle c_i^η; thus, the length of a cycle after cycle c_i^η is $|c_i^\eta|$. By contrast, a_j doubles the value of $a_j.lenCycle$ in the last round of every cycle until it starts cycle c_j^ζ. Thus, letting $lenCycle_i$ and $lenCycle_j$ be the values of $a_i.lenCycle$ and $a_j.lenCycle$ in the AgreeID and MakeGroup stages, respectively, $lenCycle_j \equiv 0 \pmod{lenCycle_i}$ holds. Hence, when a_j starts a cycle after cycle c_j^ζ, a_i also starts a cycle almost simultaneously.

Observation **5.** *Let a_i be a good agent and a_j be a good agent that starts the* AgreeID *stage not more than t_{EX} rounds later than a_i. When a_j starts a cycle*

c_j^γ, a_i *starts a cycle between rounds* $c_j^\gamma[1] - t_{EX}$ *and* $c_j^\gamma[t_{EX}]$.

Next, we prove that when an agent a_i executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$ during a cycle except for the last round, a_i meets all good agents during a core period of a cycle. If a_i starts $\texttt{REN}(\text{extendId}(a_i.id, 0))$, it stops the procedure at the middle of a cycle only if a_i finds the agent with a target ID in the MakeGroup stage or if a_i meets a reliable group after the start of the MakeCandidate stage. For a period *prd* composed of multiple rounds, we say "a_i executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$ without interruption throughout period *prd*" if a_i executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$ during period *prd*, except for the last round of α. We also call period *prd*, except for the first and last t_{EX} rounds a core period of period *prd*. Here, period α represents a cycle and the execution period of a rendezvous procedure. First, we consider the condition for two different good agents to meet.

Lemma **15.** *Let* a_i *and* a_j *be two different good agents and* prd_i *be a period comprising at least* $3 \cdot (t_{REN}(\text{extendId}(a_i.id, 0)) + 1)$ *rounds. Assume that* a_i *executes* $\texttt{REN}(\text{extendId}(a_i.id, 0))$ *without interruption throughout period* prd_i. *Agent* a_i *meets* a_j *during a core period of period* prd_i *if* a_j *stays during a core period of period* prd_i *or executes* $\texttt{REN}(\text{extendId}(a_j.id, 0))$ *or* $\texttt{REN}(\text{extendId}(a_j.guidepostId, 0))$ *for at least* $3 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ *rounds during period* prd_i.

Proof. We break down this lemma into the following cases: **Case (1)** Agent a_j stays during a core period of period prd_i; **Case (2)** Agent a_j executes $\texttt{REN}(\text{extendId}(a_j.id, 0))$ for at least $3 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ rounds during period prd_i; and **Case (3)** Agent a_j executes $\texttt{REN}(\text{extendId}(a_j.guidepostId, 0))$ for at least $3 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ rounds during period prd_i.

In **Case (1)**, by Lemma 2, a_i visits all nodes within $t_{REN}(\text{extendId}(a_i.id, 0))$ rounds from the t_{EX}-th round of period prd_i. Time t_{EX} is smaller than $t_{REN}(\text{extendId}(a_i.id, 0))$; hence, a_i meets a_j during a core period of period prd_i.

In **Case (2)**, a_i and a_j execute $\texttt{REN}(\text{extendId}(a_i.id, 0))$ and $\texttt{REN}(\text{extendId}(a_j.id, 0))$ at the same time for at least $3 \cdot (t_{REN}(\text{extendId}(\min(a_i.id, a_j.id), 0)) + 1)$ rounds immediately following the start $\texttt{REN}(\text{extendId}(a_j.id, 0))$ by a_j. Time t_{EX} is smaller than $t_{REN}(\text{extendId}(\min(a_i.id, a_j.id), 0))$; hence, by Lemma 2, a_i meets a_j during a core period of period prd_i.

In **Case (3)**, a_j executes Line 6 of Algorithm 12; thus, a_j executes Line 4 of Algorithm 12 before executing Line 6 of Algorithm 12. By contrast, a_i executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$; hence, a_i does not execute Line 4 of Algorithm 12. Therefore, $a_i.id \notin \{x_1 \mid (x_1, x_2) \in a_j.D\}$ holds, and $a_j.guidepostId \neq a_i.id$ holds. From the same discussion of **Case (2)**, a_i meets a_j during a core period of period prd_i. □

Lemma **16.** *Let a_i be a good agent in the* COLLECTID *or* MAKECANDIDATE *stage and c_i^γ be a cycle, such that a_i starts* $\texttt{REN}(\text{extendId}(a_i.id, 0))$ *in round* $c_i^\gamma[1]$. *If a_i executes* $\texttt{REN}(\text{extendId}(a_i.id, 0))$ *without interruption throughout cycle c_i^γ, a_i meets all good agents during a core period of cycle c_i^γ.*

Proof. Agent a_i satisfies Line 8 of Algorithm 8 no later than round $c_i^\gamma[1]$; hence, $|c_i^\gamma| \geq 6 \cdot (t_{\texttt{REN}}(\text{extendId}(a_i.id, 0)) + 1)$ holds. Let a_j be another good agent and c_j^ε be the first cycle of a_j such that cycle c_i^γ includes round $c_j^\varepsilon[t_{\texttt{EX}}]$. An agent updates the length of its cycle in the COLLECTID and MAKECANDIDATE stages, but not in the AGREEID and MAKEGROUP stages, therefore, c_i^γ has the longest cycle. It holds that $|c_i^\gamma| \geq |c_j^\gamma|$. By Observations 2 and 5, a_i starts cycle c_i^γ between rounds $c_j^\varepsilon[1] - t_{\texttt{EX}}$ and $c_j^\varepsilon[t_{\texttt{EX}}]$. Agent a_i also starts cycle $c_i^{\gamma+1}$ between rounds $c_j^{\varepsilon+\alpha+1}[1] - t_{\texttt{EX}}$ and $c_j^{\varepsilon+\alpha+1}[t_{\texttt{EX}}]$ for a non-negative integer α. By the behavior of `MakeReliableGroup` and from the condition of Line 4 of Algorithm 12, in each cycle, a_j executes $\texttt{REN}(\text{extendId}(a_j.id, 0))$ or $\texttt{REN}(\text{extendId}(a_j.guidepostId, 0))$ for at least $(1/2) \cdot a_j.lenCycle$ rounds or stays at the current node throughout the cycle. When a_j starts the rendezvous procedure in some cycle $c_j^{\varepsilon+\beta}$ $(0 \leq \beta \leq \alpha)$, a_j satisfies Line 8 of Algorithm 8; thus, $|c_j^{\varepsilon+\beta}| \geq 6 \cdot (t_{\texttt{REN}}(\text{extendId}(a_j.id, 0)) + 1)$ holds. Time $t_{\texttt{EX}}$ is smaller than $t_{\texttt{REN}}(\text{extendId}(a_j.id, 0))$; therefore, a_j stays during a core period of cycle c_i^γ or executes $\texttt{REN}(\text{extendId}(a_j.id, 0))$ or $\texttt{REN}(\text{extendId}(a_j.guidepostId, 0))$ for at least $3 \cdot (t_{\texttt{REN}}(\text{extendId}(a_j.id, 0)) + 1)$ rounds during cycle c_i^γ. By Lemma 15, a_i meets a_j during a core period of cycle c_i^γ. □

In the COLLECTID stage, an agent a_i executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$ without interruption throughout a cycle whose length is at least $6 \cdot (t_{\texttt{REN}}(\text{extendId}(a_i.id, 0)) + 1)$. Agent a_i meets all good agents by the end of the cycle; thus, we have the following corollary.

Corollary **1.** *For good agent a_i, if $a_i.stage \in \{MakeCandidate,$* AGREEID*, $MakeGroup\}$ holds, $a_i.S_p$ contains the IDs of all good agents; hence, $k = g + f \geq |a_i.S_p| \geq g$ holds.*

By $k \geq 8f + 7$, $g \geq 7f + 7$, and Corollary 1, we have the following corollary:

Corollary **2.** *For good agent a_i, if $a_i.stage \in \{MakeCandidate,$* AGREEID*, $MakeGroup\}$ holds, $g > (7/8)k \geq (7/8)|a_i.S_p|$ holds.*

We now consider the MAKECANDIDATE stage. Let a_{max} be the good agent with the largest ID. Regarding this stage, we clarify the two following facts.

Fact (1) All good agents finish the MAKECANDIDATE stage in some bounded rounds.

Fact (2) At least $(3/8)k$ good agents start the AGREEID stage almost simultaneously.

By $g \geq 7f + 7$, Fact (2) implies that at least $(3/8)k \geq (3/8)(8f + 7) > 3f + 1$ good agents start the AGREEID stage almost simultaneously; thus, we can claim that the behavior of the MAKECANDIDATE stage achieves the purpose of this stage. We prove Fact (1) with Lemma 17 to Lemma 19 and Fact (2) with Lemma 20 to Corollary 3.

Lemma **17.** *Let a_i be a good agent and c^{γ}_{max} be the first cycle of the* MAKECANDIDATE *stage of a_{max}. Agent a_i executes $a_i.stage \leftarrow AgreeID$ by round $c^{\gamma+1}_i[last]$.*

Proof. First, we prove that every good agent a_i executes $a_i.ready \leftarrow True$ by round $c^{\gamma}_{max}[t_{EX}]$. Agent a_{max} has the largest ID among good agents; hence, every good agent starts the MAKECANDIDATE stage before a_{max} executes the MAKECANDIDATE stage for t_{EX} rounds. We consider two cases here. The first case is where a good agent a_i is in the MAKECANDIDATE stage at the beginning of round $c^{\gamma}_{max}[t_{EX}]$. Agent a_{max} satisfies Line 8 of Algorithm 8 before cycle c^{γ}_{max}; hence, $a_{max}.lenCycle \geq 12 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ holds in cycle c^{γ}_{max}. By Observation 2, $|c^{\gamma}_i| = |c^{\gamma}_{max}|$ holds. Agent a_{max} has the largest ID among good agents; therefore, for a good agent a_j, $a_i.lenCycle \geq 12 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) \geq 12 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ holds in cycle c^{γ}_i. Variable $a_i.S_p$ contains IDs of all good agents by Corollary 1; thus, $a_i.S_p$ contains at least g IDs no larger than $a_{max}.id$. It holds that $g > (7/8)|a_i.S_p|$ by Corollary 2; therefore, a_i satisfies Line 5 of Algorithm 9 by round $c_i^\gamma[1]$. By Observation 2, a_i and a_{max} start cycles c_i^γ and c_{max}^γ almost simultaneously. Thus, a_i executes $a_i.ready \leftarrow True$ by round $c_{max}^\gamma[t_{EX}]$. The second case is where a_i finishes the MAKECANDIDATE stage before round $c_{max}^\gamma[t_{EX}]$. In this case, a_i satisfies Line 9 of Algorithm 9, that is, $|a_i.R| \geq (3/4)|a_i.S_p|$ holds. Therefore, a_i satisfies Line 6 of Algorithm 9, and a_i executes $a_i.ready \leftarrow True$ before round $c_{max}^\gamma[t_{EX}]$ at the latest.

Next, we prove that every good agent a_i executes $a_i.stage \leftarrow AgreeID$ by round $c_i^{\gamma+1}[last]$. Consider the situation that a_i has not yet executed $a_i.stage \leftarrow AgreeID$ at the beginning of round $c_i^\gamma[1]$. From the previous discussion, all good agents execute $ready \leftarrow True$ by round $c_{max}^\gamma[t_{EX}]$. Agents a_i and a_{max} start cycles c_i^γ and c_{max}^γ almost simultaneously; thus, all good agents execute $ready \leftarrow True$ by round $c_i^\gamma[t_{EX}]$. By Lemma 16, a_i meets all good agents during a core period of cycle c_i^γ; therefore, a_i has met all good agents with $ready = True$ at the end of cycle c_i^γ. Variable $a_i.R$ contains at least g IDs at the beginning of cycle $c_i^{\gamma+1}$. It holds that $g > (7/8)|a_i.S_p|$ by Corollary 2; thus, a_i satisfies Line 9 of Algorithm 9 in round $c_i^{\gamma+1}[1]$. Agent a_i executes $a_i.stage \leftarrow AgreeID$ by round $c_i^{\gamma+1}[last]$. □

In the following lemma, we calculate the maximum value of *lenCycle* of a good agent.

Lemma **18.** *For any good agent* a_i, $a_i.lenCycle$ *is less than* $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$.

Proof. Let c_{max}^γ be the first cycle of the MAKECANDIDATE stage of a_{max}. Every good agent a_i updates $a_i.lenCycle$ only in the last round of every cycle of the COLLECTID and MAKECANDIDATE stages. By Lemma 17, a_i executes $a_i.stage \leftarrow AgreeID$ by round $c_i^{\gamma+1}[last]$. Thus, every good agent, at the latest, starts the AGREEID stage at most two cycles after cycle c_i^γ. At the beginning of cycle c_{max}^γ, $a_{max}.lenCycle < 24 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ holds. By Observation 2, $|c_i^\gamma| = |c_{max}^\gamma|$ holds. Thus, $a_i.lenCycle < 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ holds at the end of cycle $c_i^{\gamma+1}$. □

Lemma **19.** *All good agents finish the* COLLECTID *stage within* $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) - T_{ini}$ *rounds right after starting*

MakeReliableGroup. *Furthermore, all good agents finish the* MakeCandidate *stage within* $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) - T_{ini}$ *rounds right after starting* *MakeReliableGroup*.

Proof. Lemma 18 implies that, for any good agent, $lenCycle = 2^{\alpha} \cdot T_{ini} < 48(t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ holds in the last cycle of the MakeCandidate stage for some integer α. Thus, the number of elapsed rounds to finish the MakeCandidate stage is less than $T_{ini} + 2 \cdot T_{ini} + \cdots + 2^{\alpha} \cdot T_{ini} < 96 \cdot (t_{REN}(a_{max}.id) + 1) - T_{ini}$. □

From now on, we will prove that at least $(3/8)k$ good agents start the AgreeID stage almost simultaneously. We will first focus on the situation where the first good agent becomes ready to transition into the AgreeID stage.

Lemma **20.** *Let* a_{ini} *be the first good agent that executes* $ready \leftarrow True$. *Agent* a_{ini} *executes* $a_{ini}.ready \leftarrow True$ *by satisfying Line 5 of Algorithm 9.*

Proof. Let c_{ini}^{γ} be the cycle in which a_{ini} executes $a_{ini}.ready \leftarrow True$. At the beginning of cycle c_{ini}^{γ}, $|a_{ini}.R| \le f$ holds because only Byzantine agents can execute $ready \leftarrow True$ before cycle c_{ini}^{γ}. It holds that $|a_{ini}.R| \le f < (1/7)g \le (1/7)|a_{ini}.S_p|$; therefore, Line 6 of Algorithm 9 is not satisfied, while Line 5 of Algorithm 9 is satisfied. □

By Lemma 20, at least one good agent a_i executes $a_i.ready \leftarrow True$ by satisfying Line 5 of Algorithm 9. In the following lemma, let c_i^{γ} be the first cycle in which a_i executes $a_i.ready \leftarrow True$. We investigate the number of good agents executing the MakeCandidate stage by round $c_i^{\gamma}[t_{EX}]$.

Lemma **21.** *Let* a_{ini} *be the first good agent that executes* $ready \leftarrow True$ *and* c_{ini}^{γ} *be the cycle, in which* a_{ini} *executes it. Let* a_i *be a good agent in the* MakeCandidate *stage in round* $c_{ini}^{\gamma}[t_{EX}]$. *At least* $(3/4)k$ *good agents start the* MakeCandidate *stage by round* $c_i^{\gamma}[t_{EX}]$.

Proof. Let f_{ini} be the number of IDs of Byzantine agents in $a_{ini}.S_p$. By Corollary 1, $a_{ini}.S_p$ contains the IDs of all good agents at the beginning of cycle c_{ini}^{γ}; hence, $|a_{ini}.S_p| = g + f_{ini}$ holds. By Observation 2, all good agents in the MakeCandidate stage start their γ-th cycles almost simultaneously, and the cycle lengths are identical. By Lemma 20, a_{ini} executes $a_{ini}.ready \leftarrow True$ in round $c_{ini}^{\gamma}[1]$ by satisfying Line 5 of Algorithm 9. This implies that at least $(7/8)|a_{ini}.S_p|$ agents in $a_{ini}.S_p$ start the MakeCandidate stage by round $c_i^{\gamma}[t_{EX}]$. Thus, at least $(7/8)|a_{ini}.S_p| - f_{ini}$ good agents start the MakeCandidate stage by round $c_i^{\gamma}[t_{EX}]$. By $f \geq f_{ini}$, $k \geq 8f + 7$, and $g \geq 7f + 7$, $(7/8)|a_{ini}.S_p| - f_{ini} = (7/8)(g + f_{ini}) - f_{ini} = (1/8)(7g - f_{ini}) \geq (1/8)(7g - f) = (1/8)(6g + g - f) \geq (1/8)(6g + 7f + 7 - f) > (3/4)(g + f) = (3/4)k$ holds. □

In the following lemma, we check R of good agents that execute the MakeCandidate stage when a good agent executes $endMakeCandidate \leftarrow True$.

Lemma **22.** *Let a_{ini} be the first good agent that executes endMakeCandidate ← True and c_{ini}^{γ} be the cycle, in which a_{ini} executes $a_{ini}.endMakeCandidate \leftarrow True$. Let a_i be a good agent in the* MakeCandidate *stage in round $c_{ini}^{\gamma}[t_{EX}]$. At the beginning of cycle c_i^{γ}, $a_i.R$ contains at least $(1/2)|a_i.S_p|$ IDs of good agents.*

Proof. At the beginning of cycle c_{ini}^{γ}, $a_{ini}.R$ contains at least $(3/4)|a_{ini}.S_p|$ IDs of agents. Thus, $a_{ini}.R$ contains at least $(3/4)|a_{ini}.S_p| - f$ IDs of good agents. It holds that $k \geq |a_{ini}.S_p| \geq g$ by Corollary 1; thus, $(3/4)|a_{ini}.S_p| - f \geq (3/4)g - f$ holds. By $g \geq 7f + 7$, $(3/4)g - f = (1/8)(4g + 2g - 8f) > (1/8)(4g + 14f - 8f) = (1/8)(4g + 6f) \geq (1/2)(g + f)$ holds. By the behavior of the CollectID and MakeCandidate stages, since an agent collects the IDs of agents with $ready = True$ during a core period of a cycle, a_{ini} has met at least $(1/2)(g + f)$ good agents with $ready = True$ at the end of round $c_{ini}^{\gamma-1}[last - t_{EX}]$. Here, let a_j be such a good agent. Agent a_j is in the MakeCandidate stage in round $c_j^{\gamma-1}[t_{EX}]$. By Observation 2, a_{ini}, a_i, and a_j start cycles $c_{ini}^{\gamma-1}$, $c_i^{\gamma-1}$, and $c_j^{\gamma-1}$ almost simultaneously. By Lemma 16, a_i meets all good agents between rounds $c_i^{\gamma-1}[t_{EX} + 1]$ and $c_i^{\gamma-1}[last - t_{EX}]$; hence, a_i meets a_j by round $c_i^{\gamma-1}[last - t_{EX}]$. By $k \geq |a_i.S_p| \geq g$, $a_i.R$ contains at least $(1/2)(g + f) \geq (1/2)|a_i.S_p|$ IDs of good agents at the beginning of cycle c_i^{γ}. □

By Lemma 17, every good agent starts the AgreeID stage by the t_{EX}-th round of the first cycle of the AgreeID stage of a_{max}. In the following lemma, we show that several good agents start the AgreeID stage almost simultaneously.

Lemma **23.** *Let a_{ini} be the first good agent that starts the* AgreeID *stage and c_{ini}^{γ} be a cycle, in which a_{ini} starts the* AgreeID *stage. At least $(3/4)k$ good* *agents start the* AgreeID *stage between rounds* $c_{ini}^{\gamma}[1]-t_{\mathtt{EX}}$ *and* $c_{ini}^{\gamma}[t_{\mathtt{EX}}]$ *or rounds* $c_{ini}^{\gamma+1}[1]-t_{\mathtt{EX}}$ *and* $c_{ini}^{\gamma+1}[t_{\mathtt{EX}}]$.

Proof. First, we prove that at least $(3/4)k$ good agents have started the MakeCandidate stage at the beginning of round $c_{ini}^{\gamma-1}[t_{\mathtt{EX}}]$. By the behavior of the MakeCandidate stage, if a_{ini} starts the AgreeID stage in cycle c_{ini}^{γ}, a_{ini} executes $a_{ini}.endMakeCandidate \leftarrow True$ in round $c_{ini}^{\gamma-1}[1]$. In other words, a_{ini} meets at least $(3/4)|a_{ini}.S_p|$ agents with $ready = True$ by round $c_{ini}^{\gamma-2}[last-t_{\mathtt{EX}}]$. By Corollary 1 and $g \geq 7f+7$, $(3/4)|a_{ini}.S_p| \geq (3/4)g \geq (3/4)(7f+7) > f$ holds. Thus, at least one good agent has executed $ready \leftarrow True$ at the end of round $c_{ini}^{\gamma-2}[last-t_{\mathtt{EX}}]$. By Lemma 20, the first good agent that executes $ready \leftarrow True$ satisfies Line 5 of Algorithm 9. Therefore, by Lemma 21, at least $(3/4)k$ good agents start the MakeCandidate stage by round $c_{ini}^{\gamma-1}[t_{\mathtt{EX}}]$.

Next, we prove this lemma. Let A_{em} be a set of good agents in the MakeCandidate stage in round $c_{ini}^{\gamma-1}[t_{\mathtt{EX}}]$. From previous discussion, $|A_{em}| \geq (3/4)k$ holds. Set A_{em} is divided into the following two sets A_1 and A_2: A_1 is a set of agents that start the AgreeID stage between rounds $c_{ini}^{\gamma}[1]-t_{\mathtt{EX}}$ and $c_{ini}^{\gamma}[t_{\mathtt{EX}}]$. Consider an arbitrary agent a_{em} in A_{em}. Agent a_{ini} executes $a_{ini}.endMakeCandidate \leftarrow True$ in cycle $c_{ini}^{\gamma-1}$; hence, by Lemma 22, $a_{em}.R$ contains at least $(1/2)|a_{em}.S_p|$ IDs of good agents at the beginning of cycle $c_{em}^{\gamma-1}$. Therefore, a_{em} satisfies Line 6 of Algorithm 9 and executes $a_{em}.ready \leftarrow True$ in round $c_{em}^{\gamma-1}[1]$. Consider an agent a_i in A_2. By Lemma 16, a_i meets all good agents in A_{em} during a core period of cycle $c_i^{\gamma-1}$. It holds that $a_{em}.ready = True$ at the beginning of round $c_{em}^{\gamma-1}[t_{\mathtt{EX}}]$; therefore, $a_i.R$ contains at least $|A_{em}| \geq (3/4)k$ IDs at the beginning of cycle c_i^{γ}. Thus, a_i satisfies Line 9 of Algorithm 9 in round $c_i^{\gamma}[1]$ and executes $a_i.stage \leftarrow AgreeID$ in round $c_i^{\gamma}[last]$. Agents in A_1 start the AgreeID stage between rounds $c_{ini}^{\gamma}[1]-t_{\mathtt{EX}}$ and $c_{ini}^{\gamma}[t_{\mathtt{EX}}]$, and agents in A_2 start that between rounds $c_{ini}^{\gamma+1}[1]-t_{\mathtt{EX}}$ and $c_{ini}^{\gamma+1}[t_{\mathtt{EX}}]$. □

By Lemma 23, we have the following corollary:

Corollary **3.** *At least* $(3/8)k$ *good agents start the* AgreeID *stage almost simultaneously.*

We now consider the AgreeID stage. We check P_p of good agents and the execution of `PCONS` for both S_p and P_p.

Lemma **24.** *Let GC be a group candidate, a_i be a good agent in GC, and c_i^γ be a cycle of the* AgreeID *stage of a_i. If a_i executes $\mathtt{REN}(\text{extendId}(a_i.id, 0))$ without interruption throughout cycle c_i^γ, a_i meets all good agents in GC during a core period of cycle c_i^γ.*

Proof. Let a_j be another good agent in GC. By Observation 3, a_i and a_j always start cycles c_i^γ and c_j^γ almost simultaneously. By Observation 4, $|c_i^\gamma| = |c_j^\gamma|$ holds. By the behavior of `MakeReliableGroup`, a_j executes $\mathtt{REN}(\text{extendId}(a_j.id, 0))$ or $\mathtt{REN}(\text{extendId}(a_j.guidepostId, 0))$ for at least $(1/2) \cdot |c_j^\gamma|$ rounds during cycle c_j^γ. Agents a_i and a_j finish the MakeCandidate stage; therefore, $|c_i^\gamma| \geq 12 \cdot (t_{\mathtt{REN}}(\text{extendId}(a_i.id, 0)) + 1)$ and $|c_j^\gamma| \geq 12 \cdot (t_{\mathtt{REN}}(\text{extendId}(a_j.id, 0)) + 1)$ holds. Agent a_j executes $\mathtt{REN}(\text{extendId}(a_j.id, 0))$ or $\mathtt{REN}(\text{extendId}(a_j.guidepostId, 0))$ for at least $6 \cdot (t_{\mathtt{REN}}(\text{extendId}(a_j.id, 0)) + 1)$ rounds during cycle c_i^γ. By Lemma 15, a_i meets a_j during a core period of cycle c_i^γ. □

During the first cycle of the AgreeID stage, an agent collects IDs of good agents in the same group candidate, but not of the other good agents. From Lemma 24, we have the following corollary:

Corollary **4.** *Let a_i be a good agent in the* AgreeID *stage and GC be a group candidate of a_i. At the beginning of the second cycle of the* AgreeID *stage, $a_i.P_p$ contains IDs of all good agents in GC, but not of the other good agents.*

Lemma **25.** *Let GC be a group candidate, a_i be a good agent in GC, and c_i^γ be the first cycle of the* AgreeID *stage of a_i. If at least $(3/8)k$ good agents belong to GC, all good agents in GC start the* MakeGroup *stage in $O(f \cdot |c_i^\gamma|)$ rounds right after round $c_i^\gamma[1]$. Furthermore, the executions of both $\mathtt{PCONS}(a_i.S_p)$ and $\mathtt{PCONS}(a_i.P_p)$ satisfy the PBC property.*

Proof. First, we show that all good agents in GC can simulate `PCONS`. To do this, Algorithm `MakeReliableGroup` satisfies all requirements in Section 4.4. Let c_i^ε be a cycle of the AgreeID stage of a_i after cycle c_i^γ and a_j be another good agent in GC. By the behavior of `MakeReliableGroup`, a_i makes a message msg_i of a phase p in round $c_i^\varepsilon[1]$ and sends msg for the other agents in GC between rounds $c_i^\varepsilon[2]$ and $c_i^\varepsilon[last - 1]$. When a_i receives a message msg_j sent by a_j at the current node between rounds $c_i^\varepsilon[2]$ and $c_i^\varepsilon[last{-}1]$, a_i records msg_j and $a_j.id$ only if $a_i.lenCycle =$

$a_j.lenCycle$ and $a_i.numCycle = a_j.numCycle$ hold. By Observations 3 and 4, a_i and a_j have the same cycle length and start cycles almost simultaneously. Thus, $a_i.lenCycle = a_j.lenCycle$ and $a_i.numCycle = a_j.numCycle$ hold during a core period of cycle c_i^{ε}. By Lemma 24, a_i and a_j meet during a core period of cycle c_i^{ε}. Therefore, without loss of generality, a_i records msg_j and $a_j.id$ before another good agent in GC executes a local computation in the next phase $p+1$. By contrast, a_i may meet a_j that executes a different cycle than a_i between rounds $c_i^{\varepsilon}[1]$ and $c_i^{\varepsilon}[t_{EX}]$ and between rounds $c_i^{\varepsilon}[last - t_{EX} + 1]$ and $c_i^{\varepsilon}[last]$. In this case, a_j sends a message msg'_j of a phase $p' \neq p$ to a_i. However, at this time, $a_i.lenCycle = a_j.lenCycle$ and $a_i.numCycle = a_j.numCycle$ do not hold. Thus, a_i ignores msg'_j. By $k \geq 8f+7$, $(3/8)k \geq (3/8)(8f+7) = 3f + 21/8 > 3f$ holds. Hence, `MakeReliableGroup` satisfies all requirements in Section 4.4.

We now prove this lemma. From the discussion of the previous paragraph, all good agents in GC can simulate `PCONS`. Thus, by Lemma 3, the executions of both `PCONS`($a_i.S_p$) and `PCONS`($a_i.P_p$) satisfy the PBC property, and a_i finishes both `PCONS`($a_i.S_p$) and `PCONS`($a_i.P_p$) in $O(f)$ cycles after cycle c_i^{γ}. Agent a_i then executes $a_i.stage \leftarrow MakeGroup$ in the last round of the cycle in which a_i finishes both `PCONS`($a_i.S_p$) and `PCONS`($a_i.P_p$). An agent does not update the cycle length in the AgreeID stage. All good agents in GC have the same cycle length and start their cycles almost simultaneously. Hence, all good agents in GC start the MakeGroup stage in $O(f \cdot |c_i^{\gamma}|)$ rounds right after round $c_i^{\gamma}[1]$. □

By Corollary 4, for a good agent a_i, at the beginning of the second cycle of the AgreeID stage, $a_i.P_p$ contains the IDs of all good agents in a group candidate of a_i, but not of the other good agents. By Lemma 25, the execution of `PCONS`($a_i.P_p$) satisfies the PBC property; hence, by Validities 1 and 2 of the PBC property, $a_i.P_c$ contains the IDs of all good agents in the group candidate of a_i, but not of the other good agents. From this discussion, Lemma 25, and Corollary 1, we have the following corollary:

Corollary **5.** *Let GC be a group candidate, a_i be a good agent in GC, and c_i^{γ} be the first cycle of a_i that all good agents in GC are in the* MakeGroup *stage at the beginning of round $c_i^{\gamma}[t_{EX}]$. If at least $(3/8)k$ good agents belong to GC, all good agents in GC have the same P_c and S_c at the beginning of round $c_i^{\gamma}[t_{EX}]$. For each good agent a_i in GC, $|a_i.S_c| \geq g$ and $|a_i.P_c| \geq (3/8)k$ hold. Furthermore,*

$a_i.S_c$ contains the IDs of all good agents, and $a_i.P_c$ contains the IDs of all good agents in GC, but not of the other good agents.

Next, we consider the MakeGroup stage. In the following two lemmas, we prove that when there exists at least one group candidate comprising at least $(3/8)k$ good agents, at least one reliable group is created.

Lemma **26.** *Let GC be a group candidate, a_{ini} be the first good agent in GC that starts the* MakeGroup *stage, and c_{ini}^{γ} be the first cycle of a_{ini} that all good agents in GC are in the* MakeGroup *stage at the beginning of round $c_{ini}^{\gamma}[t_{EX}]$. If at least $(3/8)k$ good agents belong to GC, some good agent a_i in GC executes Lines 5 and 6 of Algorithm 12 within $(f+1)\cdot|c_i^{\gamma}|$ rounds right after round $c_i^{\gamma}[1]$.*

Proof. If a_i has executed Lines 5 and 6 of Algorithm 12 before cycle c_i^{γ}, this lemma clearly holds. Therefore, we consider the case where a_i has not executed Lines 5 and 6 of Algorithm 12 before cycle c_i^{γ}.

First, let a_j be a good agent in GC. We prove that a_j decides on at least one ID of a good agent in GC as a target ID within $(f+1)$ cycles after cycle c_j^{γ}. By Corollary 5, if at least $(3/8)k$ good agents belong to GC, $|a_j.P_c| \geq (3/8)k$ holds at the beginning of round $c_j^{\gamma}[t_{EX}]$. Since $(3/8)k \geq (3/8)(8f+7) > f+1$ holds by $k \geq 8f+7$ and a_j increments $a_j.numCycle$ by one in the last round of every cycle, a_j uses different $f+1$ IDs in $a_j.P_c$ as the target IDs during $f+1$ cycles starting from cycle c_j^{γ}. Hence, the IDs include one ID of a good agent in $a_j.P_c$.

We now prove that all good agents in GC calculate the same result of $\text{target}(P_c, numCycle) = P_c[numCycle \bmod |P_c|]$ during a core period of cycle $c_i^{\gamma'}$ for any integer $\gamma' \geq \gamma$. All good agents in GC start a cycle almost simultaneously by Observation 3; therefore, all good agents in GC have the same $numCycle$ during a core period of cycle $c_i^{\gamma'}$. By Corollary 5, all good agents in GC have the same P_c at the beginning of round $c_i^{\gamma}[t_{EX}]$. Hence, all good agents in GC calculate the same result of $\text{target}(P_c, numCycle) = P_c[numCycle \bmod |P_c|]$ during a core period of cycle $c_i^{\gamma'}$.

Based on the above, all good agents in GC set the ID of the same good agent in GC as the target ID during a core period of cycle c_i^{α} for some α satisfying $\gamma \leq \alpha \leq \gamma + f$.

Next, let c_i^{ε} be the first cycle of a_i that all good agents set the ID of the same good agent in GC as a target ID during a core period of this cycle. We prove that all good agents in GC gather at a single node by round $c_i^{\varepsilon}[(1/2)\cdot|c_i^{\varepsilon}|-t_{EX}]$. Let a_{gt} be the good agent with the target ID during a core period of cycle c_i^{ε}, a_{gm} be the good agent with the largest ID of good agents in GC, and a_{gs} be a good agent in $GC \setminus \{a_{gt}\}$. By the behavior of the MakeGroup stage, a_{gt} stays at the current node during the first-subcycle of cycle c_{gt}^{ε}, and agents other than a_{gt} search for a_{gt} during the first-subcycle of their cycles. Agent a_{gm} finishes the MakeCandidate stage, and all good agents in GC have the same cycle length by Observation 4; thus, for a good agent a_j in GC, $|c_j^{\varepsilon}| \geq 12 \cdot (t_{REN}(\text{extendId}(a_{gm}.id, 0)) + 1)$ holds. By Observation 3, all good agents in GC start their cycle almost simultaneously. Thus, a_{gt} stays during a core period of the first-subcycle of cycle c_{gs}^{ε}. By Lemma 15, a_{gs} meets a_{gt} during a core period of the first-subcycle of cycle c_{gs}^{ε}, that is, by round $c_i^{\varepsilon}[(1/2)\cdot|c_i^{\varepsilon}|-t_{EX}]$.

We prove this lemma by contradiction. We assume that no good agent in GC executes Lines 5 and 6 of Algorithm 12 by the end of cycle $c_i^{\gamma+f}$. In this case, all good agents in GC gather at a single node in the round that exists during a core period of the first-subcycle of cycle c_i^{β} for some β satisfying $\gamma \leq \beta \leq \gamma + f$. Let r_g be such a round.

First, we show that, for every good agent a_j in GC, a_i stores $(a_j.id, a_j.lenCycle - a_j.numRound)$ in $a_i.D$ in round r_g. Agent a_j is in the MakeGroup stage. All good agents in GC start a cycle almost simultaneously, and round r_g exists during a core period of the first-subcycle of cycle c_i^{β}; hence, a_i and a_j are in the first-subcycle in round r_g. By Observation 4, $a_i.lenCycle = a_j.lenCycle$ holds. By Corollary 5, $a_i.S_c = a_j.S_c$ and $|a_i.S_c| \geq g$ hold. It holds that $g > (7/8)k \geq (7/8)|a_i.S_p|$ by Corollary 2; therefore, $|a_i.S_c| \geq g \geq (7/8)|a_i.S_p|$ holds. By the contradiction assumption, $a_i.numRemainRound = a_j.numRemainRound = \infty$ holds. Hence, a_i stores $(a_j.id, a_j.lenCycle - a_j.numRound)$ in $a_i.D$ in round r_g.

Next, let $D_{rr} = \{x_2 \mid (x_1, x_2) \in a_i.D\}$. We show that, for every good agent a_j in GC, when a_i stores $(a_j.id, a_j.lenCycle - a_j.numRound)$ in $a_i.D$, $\text{median}(D_{rr}) \geq (1/2)\cdot a_i.lenCycle$ holds. The number of good agents in GC is at least $(3/8)k$; thus, $|D_{rr}| \geq (3/8)k$ holds. By $k \geq 8f + 7$, $(3/8)k \geq (3/8)(8f + 7) > 2f + 2$ holds. Thus, there exists at least one value of $lenCycle - numRound$ of some good agent in GC among the smallest $f + 1$ values. Similarly, there also exists at least one value of $lenCycle - numRound$ of a different good agent in GC among the largest $f+1$ values of D_{rr}. Therefore, median(D_{rr}) exists between the minimum and maximum values of *lenCycle* − *numRound* of good agents in *GC*. Agent a_j is in the first-subcycle, and $a_i.lenCycle = a_j.lenCycle$ holds; thus, $a_j.lenCycle - a_j.numRound \geq (1/2) \cdot a_i.lenCycle$ holds. It holds that median$(D_{rr}) \geq (1/2) \cdot a_i.lenCycle$.

Finally, we lead to a contradiction. Variable $a_i.S_c$ comprises at least $(7/8)|a_i.S_p|$ IDs. The number of good agents in *GC* is at least $(3/8)k$; thus, $|a_i.D| \geq (3/8)k \geq (3/8)|a_i.S_c|$ holds. The median of D_{rr} is at least $(1/2) \cdot a_i.lenCycle$. Therefore, in round r_g, a_i satisfies Line 4 of Algorithm 12 and executes Lines 5 and 6 of Algorithm 12. This is a contradiction. □

Lemma **27.** *Let a_i be a good agent. When a_i executes Lines 5 and 6 of Algorithm 12, at least $k/7$ good agents have the same D as $a_i.D$ and store the same value as $a_i.numRemainRound$ in their numRemainRound and the same ID as $a_i.guidepostId$ in their guidepostId. Moreover, when a_i executes Line 5 of Algorithm 12, the stored value exists between the minimum and maximum values of lenCycle − numRound of good agents in a group candidate of a_i.*

Proof. First, we prove that $a_i.D$ contains at least $k/7$ IDs of good agents. By the behavior of `MakeReliableGroup`, a_i satisfies Line 4 of Algorithm 12. Since $k \geq |a_i.S_p| \geq g$ holds by Corollary 1, $|a_i.S_c| \geq (7/8)|a_i.S_p| \geq (7/8)g$ holds. Therefore, $|a_i.D| \geq (3/8)|a_i.S_c| \geq (3/8)(7/8)g = (21/64)g$ holds. There exist f Byzantine agents in the network; thus, at least $(21/64)g - f$ of them are good agents. By $g \geq 7f+7$, $(21/64)g - f > (15/49)g - f = (1/7)g + (8/49)g - f > (1/7)g + (8/49)7f - f = (1/7)(g+f)$ holds. Variable $a_i.D$ includes at least $k/7$ IDs of good agents.

We now prove the first proposition. Let a_j be a good agent in $\{x_1 \mid (x_1, x_2) \in a_i.D\} \setminus \{a_i.id\}$. Agent a_i stores $(a_j.id, a_j.lenCycle - a_j.numRound)$ in $a_i.D$ by satisfying Line 3 of Algorithm 12. In other words, a_j exists at the node with a_i, and $|a_j.S_c| \geq (7/8)|a_j.S_p|$, $a_i.lenCycle = a_j.lenCycle$, $a_i.S_c = a_j.S_c$, $a_j.stage = MakeGroup$, $a_j.numRound \leq (1/2) \cdot a_i.lenCycle$, and $a_j.numRemainRound = \infty$ hold. Agents a_i and a_j observe the same states of agents at the current node when a_i executes Line 3 of Algorithm 12. Thus, $a_i.D = a_j.D$ holds. It is established that $|a_i.D| \geq (3/8)|a_i.S_c|$, $a_i.D = a_j.D$, and $|a_i.S_c| = |a_j.S_c|$; hence, $|a_j.D| \geq (3/8)|a_j.S_c|$ holds. Since a_i satisfies Line 4 of Algorithm 12, median$(\{x_2 \mid (x_1, x_2) \in a_i.D\})$ is larger than $(1/2) \cdot a_i.lenCycle$. Because $a_i.D = a_j.D$ and $a_i.lenCycle = a_j.lenCycle$

holds, median($\{x_2 \mid (x_1, x_2) \in a_j.D\}$) is also larger than $(1/2) \cdot a_j.lenCycle$. Therefore, a_j satisfies Line 4 of Algorithm 12 and executes Lines 5 and 6 of Algorithm 12.

Finally, we prove the second proposition. Let GC be a group candidate of a_i, and set $D_{rr} = \{x_2 \mid (x_1, x_2) \in a_i.D\}$ when a_i executes Line 5 of Algorithm 12. From Line 4 of Algorithm 12, $D_{rr} \geq (3/8)|a_j.S_c|$ holds. It holds that $|a_i.S_c| \geq g$ by Corollary 5; thus, $D_{rr} \geq (3/8)g$ holds. By $g \geq 7f + 7$, $D_{rr} \geq (3/8)(7f + 7) > 2f + 2$ holds. Thus, there exists at least one value of $lenCycle - numRound$ of some good agents in GC among the smallest $f + 1$ values. Similarly, at least one value of $lenCycle - numRound$ of a different good agent exists in GC among the largest $f+1$ values of D_{rr}. Hence, median(D_{rr}) exists between the minimum and maximum values of $lenCycle - numRound$ of good agents in GC. □

Lemma **28.** *Let a_i be a good agent that executes Lines 5 and 6 of Algorithm 12 and GC' be a set of good agents in $a_i.D$. Agent a_i does not store an ID of a good agent in $a_i.BL$. When a_i stores an ID in $a_i.BL$, the other good agents in GC' also store the ID in their BL.*

Proof. By the behavior of the MakeGroup stage, let r be the round that a_i executes Lines 5 and 6 of Algorithm 12. Agent a_i executes `REN`(extendId($a_i.guidepostId, 0$)) for $a_i.numRemainRound$ rounds from round $r + 1$. By Lemma 27, all good agents in GC' store the same ID as $a_i.guidepostId$ in their *guidepostId* and the same value as $a_i.numRemainRound$ in $a_i.numRemainRound$ in round r. They executes `REN`(extendId($a_i.guidepostId, 0$)) for $a_i.numRemainRound$ rounds from round $r+1$. Thus, all good agents are at the same node while executing the rendezvous procedure with their *guidepostId*. By the behavior of `MakeReliableGroup`, an agent in the MakeGroup stage does not initialize its *numRemainRound*. Hence, the result of `detectByzantine`($A_i, a_i.D$) does not include the ID of any good agent in GC'. All good agents in GC' witness the same state of the current node from round $r + 1$; thus, this lemma holds. □

Finally, we prove the complexity of `MakeReliableGroup`.

Theorem **3.** *Let n be the number of nodes, k be the number of agents, g be the number of good agents, f be the number of weakly Byzantine agents, and a_{max} be a good agent with the largest ID among good agents. If the upper bound N of n*

is given to agents, and $k \geq 8f + 7$ *holds, Algorithm 6 makes good agents create at least one reliable group in* $O(f \cdot t_{REN}(\text{extendId}(a_{max}.id), 0))$ *rounds. All good agents in each reliable group have also formed the reliable group at the same time. Two reliable groups with the same group ID are not created within the* t_{EX} *rounds right after the first reliable group is created.*

Proof. First, we prove the first two propositions. By Lemma 19, all good agents finish the MAKECANDIDATE stage in $O(t_{REN}(\text{extendId}(a_{max}.id), 0))$ rounds right after starting `MakeReliableGroup`. By Lemma 23 and Corollary 3, in $O(t_{REN}(a_{max}.id))$ rounds right after starting `MakeReliableGroup`, it happens once that at least $(3/8)k$ good agents start the AGREEID stage almost simultaneously. Let GC be a group candidate of at least $(3/8)k$ good agents, a_i be a good agent in GC, c_i^ζ be a cycle of a_i, such that all good agents in GC start the AGREEID stage between rounds $c_i^\zeta[1] - t_{EX}$ and $c_i^\zeta[t_{EX}]$, and c_i^η be the first cycle of a_i, such that all good agents in GC are in the MAKEGROUP stage at the beginning of round $c_i^\eta[t_{EX}]$. By Lemma 25, all good agents in GC start the MAKEGROUP stage in $O(f \cdot |c_i^\zeta|)$ rounds right after round $c_i^\zeta[1]$. By Lemma 26, a_i executes Lines 5 and 6 of Algorithm 12 within $(f + 1) \cdot |c_i^\eta|$ rounds right after round $c_i^\eta[1]$. The value of $a_i.numRemainRound$ exists between the minimum and maximum values among $lenCycle - numRound$ of the good agents in GC. By Lemma 27, when a_i executes Lines 5 and 6 of Algorithm 12, at least $k/7$ good agents have the same D as $a_i.D$ and store the same ID as $a_i.guidepostId$ in their *guidepostId* and the same value as $a_i.numRemainRound$ in their *numRemainRound*. By the behavior of the MAKEGROUP stage, let r be the round that a_i executes Lines 5 and 6 of Algorithm 12. Agent a_i executes `REN`$(\text{extendId}(a_i.guidepostId, 0))$ for $a_i.numRemainRound$ rounds from round $r + 1$. Thus, at least $k/7$ good agents execute `REN`$(\text{extendId}(a_i.guidepostId, 0))$ for $a_i.numRemainRound$ rounds from round $r + 1$. By Lemma 28, when some good agents in GC execute `REN`$(\text{extendId}(guidepostId, 0))$, they do not store each other's ID in BL and always have the same BL. Therefore, at least $k/7$ good agents finish the execution of `REN`$(\text{extendId}(a_i.guidepostId, 0))$ and store the same group ID in their *gid* at the same time within $(f + 1) \cdot |c_i^\eta| + t_{EX}$ rounds right after round $c_i^\eta[1]$. The maximum length of the cycles is at most $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ by Lemma 18; thus, a reliable group is created in $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) \cdot O(f) + 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)(f + 1) + t_{EX} = O(f \cdot t_{REN}(\text{extendId}(a_{max}.id, 0)))$ rounds right after starting cycle c_i^{ζ}. Round $c_i^{\zeta}[1]$ is in $O(t_{REN}(\text{extendId}(a_{max}.id, 0)))$ rounds right after starting `MakeReliableGroup`; therefore, a reliable group is created in $O(t_{REN}(\text{extendId}(a_{max}.id, 0))) + O(f \cdot t_{REN}(\text{extendId}(a_{max}.id, 0))) = O(f \cdot t_{REN}(\text{extendId}(a_{max}.id, 0)))$ after starting `MakeReliableGroup`.

Finally, we prove the last proposition by contradiction. Let r_{ini} be the first round when a reliable group is created, a_i be a good agent in a reliable group created within t_{EX} rounds from round r_{ini}, and a_j be a good agent in another reliable group created within the t_{EX} rounds from round r_{ini}. Assume that a_i and a_j set the same group ID gid' as a group ID. Let a_ℓ be an agent with gid', and sets $D_i^{id} = \{x_1 \mid (x_1, x_2) \in a_i.D\}$ and $BL_i = a_i.BL$ (resp. sets $D_j^{id} = \{x_1 \mid (x_1, x_2) \in a_j.D\}$ and $BL_j = a_j.BL$) when a_i (resp. a_j) executes Line 18 of Algorithm 12, that is, it becomes a member of a reliable group. By the contradiction assumption, $\min(D_i^{id} \setminus BL_i) = \min(D_j^{id} \setminus BL_j) = gid'$ holds; thus, $gid' \in D_i^{id}$, $gid' \in D_j^{id}$, $gid' \notin BL_i$, and $gid' \notin BL_j$ hold. By contrast, for $gid' \in D_i^{id}$ and $gid' \in D_j^{id}$ to hold, a_i, a_j, and a_ℓ must exist at the same node when a_i and a_j update their D or a_j (resp. a_i) stores gid' in $a_j.D$ (resp. $a_i.D$) after a_i (resp. a_j) stores gid' in $a_i.D$ (resp. $a_j.D$). First, we consider the case where a_i, a_j, and a_ℓ exist at the same node when a_i and a_j update their D. Agents a_i and a_j belong to different reliable groups; hence, a_i and a_j do not update their D at the same nodes in the same rounds unless $a_i.lenCycle \neq a_j.lenCycle$. In the case where $a_i.lenCycle \neq a_j.lenCycle$ holds, either $a_i.lenCycle \neq a_\ell.lenCycle$ or $a_j.lenCycle \neq a_\ell.lenCycle$ holds, and either $gid' \notin D_i^{id}$ or $gid' \notin D_j^{id}$ holds. Next, without loss of generality, we consider the case where a_j stores gid' in $a_j.D$ after a_i stores gid' in $a_i.D$. In this case, a_j finishes the MakeCandidate stage; therefore, $a_j.lenCycle \geq 12 \cdot t_{REN}(\text{extendId}(a_j.id, 0))$. By Line 4 of Algorithm 12, the last update round of $a_j.D$ is at least $6 \cdot t_{REN}(\text{extendId}(a_j.id, 0))$ rounds before a_j decides a target ID. Time t_{EX} is smaller than $t_{REN}(\text{extendId}(a_j.id, 0))$; therefore, a_j cannot store gid' in $a_j.D$ after a_i decides a group ID. Before a_i decides a group ID, a_ℓ must participate in the event when a_j updates $a_j.D$. To participate in the event, a_ℓ moves away from a_i or initializes $a_\ell.numRemainRound$, allowing a_i to store gid' in $a_i.BL$. Hence, either $gid' \notin D_i^{id}$ or $gid' \notin D_j^{id}$ holds, or either $gid' \in BL_i$ or

$gid' \in BL_j$ holds. Thus, this is a contradiction. □

2.3.2 Gathering with Non-simultaneous Termination

Next, we prove that all good agents gather at the same node and transition into a terminal state. By the behavior of `ByzantineGathering`, an agent executes `MakeReliableGroup` until the current node contains a reliable group. By Theorem 3, agents create at least one reliable group after starting `ByzantineGathering`. Therefore, we prove that after a reliable group is created, all good agents gather at a single node using a reliable group and transition into a terminal state. Hereinafter, we say "two agents or two reliable groups start rendezvous procedures with their group IDs almost simultaneously" if they start the rendezvous procedures within the t_{EX} rounds. We say "an agent and a reliable group start a cycle and a rendezvous procedure with its group ID almost simultaneously" if they start the cycle and the rendezvous procedure within the t_{EX} rounds.

First, we focus on the start round and the execution period of a rendezvous procedure by a good agent a_i in a reliable group. Let GC be a group candidate of a_i and c_i^γ be the cycle when a_i executes $\texttt{REN}(\text{extendId}(a_i.\mathit{guidepostId}, 0))$. Recall that $|c_i^\gamma|$ is the value of $a_i.\mathit{lenCycle}$ at the beginning of cycle c_i^γ; thus, round $c_i^\gamma[\mathit{last}]$ is round $c_i^\gamma[1] + a_i.\mathit{lenCycle} - 1$. We refer to the period from round $c_i^\gamma[\mathit{last}] + 1$ to $c_i^\gamma[\mathit{last}] + a_i.\mathit{lenCycle}$ as an "additional cycle" and simply write it as $c_i^{\gamma+1}$. Intuitively, cycle $c_i^{\gamma+1}$ is the next cycle if a_i had not executed $\texttt{REN}(\text{extendId}(a_i.\mathit{guidepostId}, 0))$ in cycle c_i^γ. Recall that a_i may not finish cycle c_i^γ in round $c_i^\gamma[\mathit{last}]$. Thus, the start of cycle $c_i^{\gamma+1}$ is not necessarily the same as the start of $\texttt{REN}(\text{extendId}(a_i.\mathit{gid}, 1))$. By the behavior of `MakeReliableGroup`, a_i executes Lines 5 and 6 of Algorithm 12 before storing a group ID in $a_i.\mathit{gid}$. By Lemma 27, when a_i executes Line 5 of Algorithm 12, the stored value exists between the minimum and maximum values of $\mathit{lenCycle} - \mathit{numRound}$ of the good agents in GC. Thus, when a_i starts $\texttt{REN}(\text{extendId}(a_i.\mathit{gid}, 1))$, the start round exists between the earliest and latest in rounds $\{c_j^{\gamma+1}[1] \mid a_j \in GC\}$. Therefore, we can expand the discussion of Observation 5 for the execution of $\texttt{REN}(\text{extendId}(a_i.\mathit{gid}, 1))$. More concretely, we assume that a_i starts $\texttt{REN}(\text{extendId}(a_i.\mathit{gid}, 1))$ in a round r. Let a_j be a good agent not in any reliable group in round $c_i^{\gamma+1}[t_{EX}]$ and c_j^ε

be the cycle of a_j that includes round $c_i^{\gamma+1}[t_{EX}]$. We can obtain the following: if $a_i.lenCycle \le a_j.lenCycle$ holds in round $r + t_{EX}$, cycle c_j^{ε} completely includes a core period of the execution period of $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$; otherwise, the execution period of $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$ completely includes a core period of cycle c_j^{ε}. By the behavior of `ByzantineGathering`, a_i does not update $a_i.lenCycle$ after becoming a member of a reliable group. Thus, the length of the execution of $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$ is the same as that of the last cycle in the MAKEGROUP stage of a_i. The following observation summarizes this discussion:

Observation **6.** *Let a_i be a good agent in a reliable group and r be the round that a_i starts $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$. Let a_j be a good agent not in any reliable group in round $r + t_{EX}$ and c_j^{ε} be the cycle of a_j that includes round $r + t_{EX}$. If $a_i.lenCycle \le a_j.lenCycle$ holds in round $r + t_{EX}$, cycle c_j^{ε} completely includes a core period of the execution period of $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$; otherwise, the execution period of $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$ completely includes a core period of cycle c_j^{ε}. Furthermore, the length of the execution of $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$ is the same as that of $|c_i^{\gamma}|$.*

Next, we focus on the execution of a rendezvous procedure by a reliable group RG of a_i. By Definition 3, all good agents in GC have the same group ID. By Theorem 3, RG is created at the same time. By the behavior of `MakeReliableGroup`, RG is composed of good agents with a D containing each other's ID; thus, all good agents in RG have the same value of *lenCycle*. Every good agent in RG executes a rendezvous procedure with the same group ID for the same period until they meet a reliable group with a smaller group ID. The following observation formally shows this fact:

Observation **7.** *Let a_i be a good agent in a reliable group and RG be a reliable group of a_i. All good agents in RG execute $\mathtt{REN}(\text{extendId}(a_i.gid, 1))$ for $a_i.lenCycle$ rounds from the next round when RG is created as long as they do not meet a reliable group with a smaller group ID.*

We now focus on how a reliable group appears from other agents. Assume that a good agent a_i has started the MAKECANDIDATE stage. By Definition 3, a reliable group contains at least $k/7$ good agents, and their group IDs are identical. By Observation 7, all good agents in any reliable group execute a

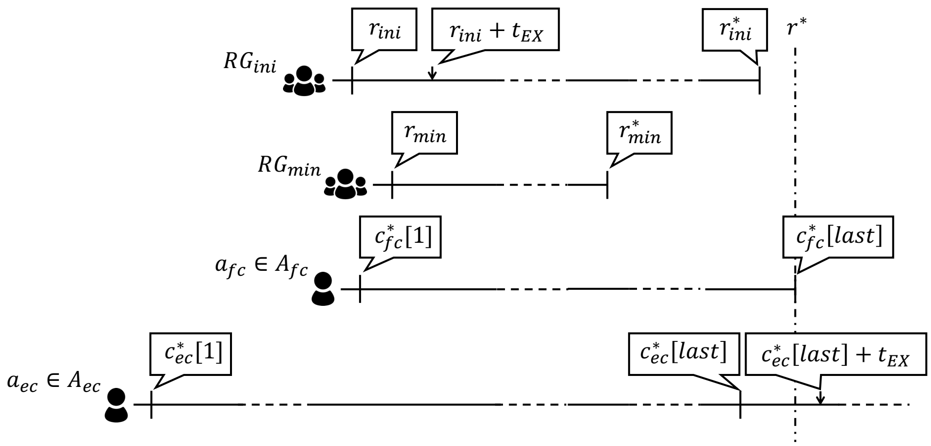

Figure 5. Example of the execution of rendezvous procedures by reliable groups RG and RG_{min} and cycles by good agents a_{fc} and a_{ec}.

rendezvous procedure with the same group ID for the same period from the next round when the reliable group is created; thus, they are always at the same node. By Corollary 1, $k \geq |a_i.S_p| \geq g$ holds; hence, $k/7 \geq |a_i.S_p|/7 \geq g/7$ holds. By $g \geq 7f + 7$, $|a_i.S_p|/7 \geq g/7 \geq (1/7)(7f + 7) = f + 1$ holds. Thus, when a_i meets at least $(1/7)|a_i.S_p|$ agents with the same group ID, a_i understands that at least one good agent exists among them and decides that they are trustworthy. Hence, when a_i meets a reliable group, a_i recognizes the group as a reliable group. By contrast, because $|a_i.S_p|/7 \geq f + 1$ holds, only f Byzantine agents are not enough to form a reliable group. Thus, if a group of only f Byzantine agents exists at the node with a_i, a_i does not recognize the group as a reliable group. The following observation formally shows this fact:

Observation **8.** *Assume that a good agent a_i has started the* MakeCandidate *stage. If a_i meets a reliable group, a_i recognizes the group as a reliable group; otherwise, a_i does not recognize the group as a reliable group.*

We next focus on good agents in the CollectID stage after a reliable group is created. We visualize the variables defined in the rest of this paragraph in Fig. 5. In this figure, the spacing between two vertical bars represents one cycle or the execution of a rendezvous procedure with a group ID. Let RG_{ini} be the first reliable group that starts a rendezvous procedure, r_{ini} be the first round of the execution of the rendezvous procedure of RG_{ini}, and SRG_{ini} be a set of reliable groups that start the rendezvous procedures almost simultaneously as RG_{ini}. Furthermore, let RG_{min} be the reliable group with the smallest group ID among SRG_{min} and r_{min} be the round, in which RG_{min} starts a rendezvous procedure. By assumption of SRG_{min}, r_{min} exists between rounds r_{ini} and $r_{ini} + t_{\mathtt{EX}}$. Let A_{ec} be a set of good agents in the CollectID stage in round $r_{ini} + t_{\mathtt{EX}}$ and A_{fc} be a set of good agents neither in A_{ec} nor in any reliable group of SRG_{ini}. For a good agent $a_i \in A_{ec} \cup A_{fc}$, let c_i^* be the cycle of a_i that includes round $r_{ini} + t_{\mathtt{EX}}$. For a good agent a_j in a reliable group of SRG_{ini}, let r_j^* be the round that is $t_{\mathtt{REN}}(extendId(a_j.gid, 1))$ rounds after a_j executes Lines 10–14 of Algorithm 13 for the first time, that is, after a_j starts $\mathtt{REN}(extendId(a_j.gid, 1))$. Let round $r^* = \max(\{c_i^*[last] \mid a_i \in A_{ec} \cup A_{fc}\} \cup \{r_j^* \mid a_j$ be a good agent in a reliable group of $SRG_{ini}\})$.

Consider a case where A_{ec} is not empty. By the behavior of `MakeReliableGroup`, an agent extends the length of its cycle in the CollectID or MakeCandidate stage, but not in the AgreeID or MakeGroup stages; therefore, good agents in the CollectID or MakeCandidate stage have the longest cycles. By Observation 5, cycle c_{ec}^* of a good agent a_{ec} in A_{ec} completely includes a core period of cycle c_{fc}^* of a good agent a_{fc} in A_{fc}. By Observation 6, cycle c_{ec}^* also completely includes the execution period of $\mathtt{REN}(\mathrm{extendId}(a_j.gid, 1))$, except for the first and last $t_{\mathtt{EX}}$ rounds. Hence, if A_{ec} is not empty, round r^* exists between rounds $c_{ec}^*[last]$ and $c_{ec}^*[last] + t_{\mathtt{EX}}$. From this discussion, we have the following observation:

Observation **9.** *If A_{ec} is not empty, for the cycle c_i^* of a good agent in A_{ec}, r^* exists between rounds $c_i^*[last]$ and $c_i^*[last] + t_{\mathtt{EX}}$.*

Let $lenCycle_{rg}$ be the value of $lenCycle$ of a good agent in RG_{min} when the agent starts a rendezvous procedure with a group ID. We will now prove that RG_{min} and other good agents meet and transition into terminal states together. To do this, we must first consider the condition for a good agent in RG_{min} and a good agent not in any reliable group to meet.

Lemma **29.** *Let a_i be a good agent in RG_{min} and a_j be a good agent not in any reliable group. Let prd be the period, such that a_i executes $\mathtt{REN}(\mathrm{extendId}(a_i.gid, 1))$ without interruption, and a_j executes*

REN(extendId($a_j.id, 0$)) *or* *REN*(extendId($a_j.guidepostId, 0$)) *without interruption during this period. If* $prd \geq t_{REN}(\text{extendId}(\min(a_i.id, a_j.id), 0))$ *holds,* a_i *and* a_j *meet within the* $t_{REN}(\text{extendId}(\min(a_i.id, a_j.id), 0))$ *rounds from the first round of the period prd.*

Proof. By the characteristic of extendId, even if a_i and a_j use the same ID as the input of a rendezvous procedure, the extended IDs of a_i and a_j are different. From the selection method of *gid*, $a_i.id \geq a_i.gid$ holds. From the selection method of *guidepostId*, $a_j.id \geq a_j.guidepostId$ holds. Thus, by Lemma 2, a_i meets a_j within the $t_{REN}(\text{extendId}(\min(a_i.gid, a_j.guidepostId), 0)) \leq t_{REN}(\text{extendId}(\min(a_i.id, a_j.id), 0))$ rounds from the first round of period *prd*. □

We will now prove that RG_{min} and a good agent either in A_{fc} or in a reliable group of SRG_{ini} meet and transition into terminal states together. In the following lemma, we prove that a good agent $a_i \in A_{fc}$ and RG_{min} meet if both a_i and RG_{min} do not accompany another reliable group.

Lemma **30.** *Let us assume that every good agent* a_i *in* A_{fc} *executes* `MakeReliableGroup` *until round* $c_i^*[last]$, *and every good agent* a_j *in* RG_{min} *executes* *REN*(extendId($a_j.gid, 1$)) *for* $lenCycle_{rg}$ *rounds without interruption from round* r_{min}. *If* a_i *is in the* MakeCandidate *or the* AgreeID *stage,* a_i *and* RG_{min} *meet by round* $\min(r_{min} + lenCycle_{rg}, c_i^*[last - t_{EX}])$. *If* a_i *is in the* MakeGroup *stage,* a_i *and* RG_{min} *meet by round* $c_i^*[last - t_{EX}]$.

Proof. By Observation 7, all good agents in RG_{min} execute `REN`(extendId($a_j.gid, 1$)) for the $lenCycle_{rg}$ rounds from round r_{min} until they meet a reliable group with a smaller group ID. Therefore, by lemma assumption, to prove this lemma, it is enough for a_i and a_j to meet by round $r_{min} + lenCycle_{rg}$ or $c_i^*[last - t_{EX}]$. We consider two cases, that is, $|c_i^*| \leq lenCycle_{rg}$ and $|c_i^*| > lenCycle_{rg}$.

First, we consider the case $|c_i^*| \leq lenCycle_{rg}$. By Observation 6, the execution period of `REN`(extendId($a_j.gid, 1$)) completely includes a core period of cycle c_i^*. By the behavior of `MakeReliableGroup` and Line 4 of Algorithm 12, a_i executes `REN`(extendId($a_i.id, 0$)) or `REN`(extendId($a_i.guidepostId, 0$)) without interruption for at least $(1/2) \cdot |c_i^*| - t_{EX}$ rounds from round $c_i^*[(1/2) \cdot |c_i^*| + 1]$ at the latest. Agent a_i finishes the CollectID stage; hence, $|c_i^*| \geq 12 \cdot (t_{REN}(\text{extendId}(a_i.id, 0)) + 1)$

holds. Time t_{EX} is smaller than $t_{REN}(\text{extendId}(a_i.id, 0))$; therefore, a_i executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$ or $\texttt{REN}(\text{extendId}(a_i.guidepostId, 0))$ without interruption for at least $5 \cdot (t_{REN}(\text{extendId}(a_i.id, 0)) + 1)$ rounds during the execution period of $\texttt{REN}(\text{extendId}(a_j.gid, 1))$. Therefore, by Lemma 29, a_i and a_j meet within $t_{REN}(\text{extendId}(\min(a_i.id, a_j.id), 0))$ rounds from the first round of the rendezvous execution of a_i that overlaps with the execution period of $\texttt{REN}(\text{extendId}(a_j.gid, 1))$. That is, a_j meets a_i by round $c_i^*[last - t_{EX}] \le r_{min} + lenCycle_{rg}$.

Next, we consider the case $|c_i^*| > lenCycle_{rg}$. By Observation 6, cycle c_i^* completely includes a core period of the execution period of $\texttt{REN}(\text{extendId}(a_j.gid, 1))$. By the behavior of `MakeReliableGroup`, we break this case down into the two following cases: **Case (1)** a_i executes $\texttt{REN}(\text{extendId}(a_i.id, 0))$ or $\texttt{REN}(\text{extendId}(a_i.guidepostId, 0))$ without interruption from round r_{min} by round $c_i^*[last - t_{EX}]$; and **Case (2)** a_i may be interrupted while executing $\texttt{REN}(\text{extendId}(a_i.id, 0))$ from round r_{min} by round $c_i^*[last - t_{EX}]$).

In **Case (1)**, a_j finishes the CollectID stage; thus, $lenCycle_{rg} \ge 12 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ holds. Because t_{EX} is smaller than $t_{REN}(\text{extendId}(a_j.id, 0))$, a_j executes $\texttt{REN}(\text{extendId}(a_j.gid, 1))$ without interruption for at least $10 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ rounds during cycle c_i^*. By Lemma 29, a_i and a_j meet within the $t_{REN}(\text{extendId}(\min(a_i.id, a_j.id), 0))$ rounds from the first round of the rendezvous execution of a_j that overlaps with cycle c_i^*, that is, by round $r_{min} + lenCycle_{rg}$. It holds that $r_{min} + lenCycle_{rg} \le c_i^*[last] + t_{EX}$, and t_{EX} is smaller than $t_{REN}(\text{extendId}(a_j.id, 0))$; therefore, round $c_i^*[last - t_{EX}] = c_i^*[last] + t_{EX} - 2t_{EX}$ may exist before round $r_{min} + lenCycle_{rg}$. Thus, a_i meets a_j by round $\min(r_{min} + lenCycle_{rg}, c_i^*[last - t_{EX}])$.

In **Case (2)**, a_i is in the MakeGroup stage; thus, a_i is interrupted while executing $\texttt{REN}(\text{extendId}(a_i.id, 0))$ during the first-subcycle. Agent a_i starts $\texttt{REN}(\text{extendId}(a_i.id, 0))$ or $\texttt{REN}(\text{extendId}(a_i.guidepostId, 0))$ from round $c_i^*[(1/2) \cdot |c_i^*| + 1]$ at the latest and executes that until round $c_i^*[last - t_{EX}]$. By contrast, because an agent extends the length of its cycle by doubling that length, $(1/2) \cdot |c_i^*| \ge lenCycle_{rg}$ and $(1/2) \cdot |c_i^*| \equiv 0 (\bmod\, lenCycle_{rg})$ hold. Thus, a_j transitions into a terminal state and stays from round $c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$ at the latest. In other words, a_j remains during a core period of the second-subcycle of cycle

c_i^*. By Lemma 29, a_i meets a_j by round $c_i^*[last - t_{\mathtt{EX}}]$. □

For an agent a_i and a reliable group RG, we say "a_i follows RG in round r" if a_i and RG satisfy the following condition: (1) if all good agents in RG terminate in round r, a_i also terminates in round r; and (2) if all good agents in RG move to node v, a_i also moves to node v. In the following lemma, we prove that if a good agent finishes the CollectID stage and meets a reliable group, the good agent follows the reliable group with the smallest group ID at the node.

Lemma **31.** *Assume that at least one reliable group exists at a node v in round r'. Let RG' be the reliable group with the smallest group ID at v in round r' and a_i be a good agent that has started the* MakeCandidate *stage in round r'. Agent a_i follows RG' in round r'.*

Proof. Let C' be a set of all the group IDs of reliable groups at v in round r'. By Observation 8, a_i recognizes all reliable groups at v. Thus, a_i executes $a_i.S_{gid} \leftarrow C'$, and a_i executes $a_i.minGID \leftarrow \min(a_i.S_{gid})$. This implies that a_i executes $a_i.minGID \leftarrow \min(C')$. By the assumption of this lemma, $a_i.S_{gid} \neq \emptyset$ and $a_i.gid > a_i.minGID$ hold in round r'; thus, a_i calculates $a_i.S_{rg}$ and executes $\mathtt{FOLLOW}(a_i.S_{rg})$. Note that $a_i.S_{rg}$ contains all good agents in RG'. By Observation 7, all good agents in RG' make the same behavior. By Definition 3 and $k \geq 8f+7$, the number of good agents in RG' is at least $f+1$. Therefore, the majority of agents in RG' are good agents. Correspondingly, a_i follows RG'. □

In the following lemma, we prove that agents do not create a reliable group between rounds $r_{ini} + t_{\mathtt{EX}}$ and r^*.

Lemma **32.** *Assume that every good agent a_i in RG_{min} executes* $\mathtt{REN}$(extendId($a_i.gid$, 1)) *for the $lenCycle_{rg}$ rounds from round r_{min} without interruption. No reliable group is created between rounds $r_{ini} + t_{\mathtt{EX}}$ and r^*.*

Proof. We prove this lemma by breaking it down into the following cases: agents in A_{ec} and agents in A_{fc}. First, we consider an agent a_{ec} in A_{ec}. By Observation 9, a_{ec} starts the MakeCandidate stage from round $r^* - t_{\mathtt{EX}} + 1$ at the earliest. From Line 8 of Algorithm 8, when a_{ec} starts the MakeCandidate stage, the cycle length is sufficiently longer than $t_{\mathtt{EX}}$ rounds. Thus, a_{ec} cannot participate in the creation of a reliable group until round r^*. Next, we consider an agent a_{fc} in A_{fc}. By Lemma 30, a_{fc} and RG_{min} meet by round $c^*_{fc}[last - t_{EX}]$. By Lemma 31, if a_{fc} meets a reliable group, a_{fc} follows the reliable group. Hence, a_{fc} cannot participate in the creation of a reliable group. □

In the following lemma, we prove that good agents in RG_{min} are never interrupted while executing `REN`.

Lemma **33.** *Every good agent a_i in RG_{min} executes* `REN`(extendId($a_i.gid, 1$)) *for the $lenCycle_{rg}$ rounds without interruption from round r_{min}.*

Proof. By the behavior of `ByzantineGathering`, a_i is interrupted while executing `REN`(extendId($a_i.gid, 1$)) if it meets a reliable group with a smaller group ID. However, by Lemma 32, no reliable group is created between rounds $r_{ini} + t_{EX}$ and r^*. In other words, since $r_{ini} \leq r_{min} \leq r_{ini} + t_{EX}$ and $r_{min} + lenCycle_{rg} \leq r^*$ hold, a_i executes `REN`($a_i.gid$) for $lenCycle_{rg}$ rounds without interruption. □

By Observation 7 and Lemma 33, we obtain the following corollary:

Corollary **6.** *All good agents in RG_{min} transition into terminal states at the same node at the same time in $r_{min} + lenCycle_{rg}$.*

In the following lemma, we prove that two reliable groups in SRG_{ini} meet:

Lemma **34.** *Let RG be a reliable group in $SRG_{ini} \setminus \{RG_{min}\}$. All good agents in RG meet a reliable group with a group ID smaller than RG by round $r_{min} + lenCycle_{rg}$.*

Proof. For contradiction, we assume that some good agents in RG have never met a reliable group with a group ID smaller than RG by round $r_{min} + lenCycle_{rg}$. Let a_i be this agent and $lenCycle_i$ be the value of $a_i.lenCycle$ in the next round when RG is created. In this case, a_i executes `REN`(extendId($a_i.gid, 1$)) for the $lenCycle_i$ rounds without interruption. By Lemma 32, a_i starts `REN`(extendId($a_i.gid, 1$)) between rounds r_{ini} and $r_{ini} + t_{EX}$. Let a_j be a good agent in RG_{min}. By Lemma 33, a_j executes `REN`(extendId($a_j.gid, 1$)) for the $lenCycle_{rg}$ rounds without interruption from round r_{min}. It also holds that $r_{ini} \leq r_{min} \leq r_{ini} + t_{EX}$. Agents a_i and a_j finish the CollectID stage, and $a_i.id \geq a_i.gid$ and $a_j.id \geq$

$a_j.gid$ holds; therefore, $lenCycle_i \geq 12 \cdot (t_{\texttt{REN}}(\text{extendId}(a_i.id, 0)) + 1) \geq 12 \cdot (t_{\texttt{REN}}(\text{extendId}(a_i.gid, 1)) + 1)$ and $lenCycle_{rg} \geq 12 \cdot (t_{\texttt{REN}}(\text{extendId}(a_j.id, 0)) + 1) \geq 12 \cdot (t_{\texttt{REN}}(\text{extendId}(a_j.gid, 1)) + 1)$ hold. It holds that $a_i.gid > a_j.gid$; thus, a_i and a_j execute $\texttt{REN}(\text{extendId}(a_i.gid, 1))$ and $\texttt{REN}(\text{extendId}(a_j.gid, 1))$ at the same time for at least $t_{\texttt{REN}}(\text{extendId}(a_j.gid, 1))$ rounds from round r_{min}. By Theorem 3, every reliable group in SRG_{ini} has a different group ID. Thus, by Lemma 2, a_i and a_j meet within the $t_{\texttt{REN}}(\text{extendId}(a_j.gid, 1))$ rounds from round r_{min}, that is, by round $r_{min} + lenCycle_{rg}$. All good agents in RG_{min} stay at the same node by Observation 7, enabling a_i to meet RG_{min} by round $r_{min} + lenCycle_{rg}$. This is a contradiction. □

In the following lemma, we prove that RG_{min} and a good agent neither in RG_{min} nor in A_{ec} meet and transition into a terminal state at the same node by round r^*.

Lemma **35.** *Let a_i be a good agent that is in A_{fc} and in the* MakeCandidate *or* AgreeID *stage or a good agent in a reliable group of $SRG_{ini} \setminus \{RG_{min}\}$. Let a_j be a good agent that is in A_{fc} and in the* MakeGroup *stage. Agent a_i meets RG_{min} and transitions into a terminal state together with RG_{min} by round $r_{min} + lenCycle_{rg}$. Agent a_j also meets RG_{min} and transitions into a terminal state together with RG_{min} by round $\max(r_{min} + lenCycle_{rg}, c_j^*[last - t_{\texttt{EX}}])$.*

Proof. Let a_ℓ be a good agent in RG_{min}. By Lemma 33, all good agents in RG_{min} execute $\texttt{REN}(\text{extendId}(a_\ell.gid, 1))$ for the $lenCycle_{rg}$ rounds without interruption from round r_{min}. By Corollary 6, all good agents in RG_{min} transition into a terminal state at the same node at the same time in $r_{min} + lenCycle_{rg}$. By Lemma 31, if a good agent finishes the CollectID stage and meets RG_{min}, the agent follows RG_{min} after that. Hence, it is sufficient to prove that a_i and a_j meet RG_{min} by round $r_{min} + lenCycle_{rg}$ and round $\max(r_{min} + lenCycle_{rg}, c_j^*[last - t_{\texttt{EX}}])$, respectively. To do this, we break it down into the following cases: **Case (1)** $a_i \in RG$ for a reliable group $RG \in SRG_{ini} \setminus \{RG_{min}\}$; **Case (2)** $a_i \in A_{fc}$; and **Case (3)** a_j.

First, we consider **Case (1)**. By Lemma 34, a_i meets a reliable group RG' with a group ID smaller than RG by round $r_{min} + lenCycle_{rg}$. If RG' is RG_{min}, a_i meets RG_{min} by round $r_{min} + lenCycle_{rg}$; otherwise, a_i follows RG' by Lemma 31.

Similarly, good agents in RG' meet another reliable group RG'' with a smaller group ID by round $r_{min} + lenCycle_{rg}$. Hence, a_i also meets RG''. By repeating this discussion, a_i eventually meets RG_{min} by round $r_{min} + lenCycle_{rg}$.

Next, we consider **cases (2)** and **(3)**. By Lemma 30, a_i and a_j meet RG_{min} by rounds $\min(r_{min} + lenCycle_{rg}, c_i^*[last - t_{\mathtt{EX}}])$ and $c_j^*[last - t_{\mathtt{EX}}]$, respectively. If a_i and a_j meet a reliable group other than RG_{min} by round $r_{min} + lenCycle_{rg}$, they follow the reliable group by Lemma 31. In this case, similar to the previous paragraph, they meet RG_{min} by round $r_{min} + lenCycle_{rg}$. □

We will now prove that a good agent in A_{ec} meets RG_{min} and transitions into a terminal state together with RG_{min}. In the following lemma, for a good agent a_i in A_{ec}, we consider the behavior of a good agent not in A_{ec} based on where round r_{ini} is located in cycle c_i^*.

Lemma **36.** *Let a_i be an agent in A_{ec} and a_j be a good agent either in A_{fc} or in a reliable group of SRG_{ini}. If $r_{ini} \leq c_i^*[(1/2) \cdot |c_i^*| - t_{\mathtt{EX}}]$ holds, a_j transitions into a terminal state together with RG_{min} by round $c_i^*[(1/2) \cdot |c_i^*| + t_{\mathtt{EX}}]$; otherwise, a_j executes $\mathtt{REN}(\text{extendId}(a_j.id, 0))$ or $\mathtt{REN}(\text{extendId}(a_j.guidepostId, 0))$ for at least $t_{\mathtt{REN}}(\text{extendId}(a_j.id, 0))$ rounds without interruption from round $c_i^*[1]$ to round $c_i^*[(1/2) \cdot |c_i^*| - t_{\mathtt{EX}}]$.*

Proof. Because a_j finishes the CollectID stage, $a_j.lenCycle \geq 12 \cdot (t_{\mathtt{REN}}(\text{extendId}(a_j.id, 0)) + 1)$. To prove this lemma, we break a_j down into the following cases: **Case (1)** Agent a_j is a good agent in RG_{min}; **Case (2)** Agent a_j is a good agent that is in A_{fc} and in the MakeGroup stage; and **Case (3)** Agent a_j is a good agent that is in A_{fc} and in the MakeCandidate or AgreeID stage or is a good agent in a reliable group of $SRG_{ini} \setminus \{RG_{min}\}$.

First, we consider the state of a_j in **cases (1)** and **(2)** when a_i starts cycle c_i^*. By the behavior of `ByzantineGathering`, an agent extends the length of its cycle only in the CollectID and MakeCandidate stages; thus, a_i has the longest cycle in cycle c_i^*. Therefore, by Observation 6, cycle c_i^* completely includes core periods of cycle c_j^* and the execution period of $\mathtt{REN}(\text{extendId}(a_j.gid, 1))$. By Lemma 27, when a_j executes Line 5 of Algorithm 12, the stored value exists between the minimum and maximum values of $lenCycle - numRound$ of the good agents in a group candidate of a_j. Thus, when a_j starts $\mathtt{REN}(\text{extendId}(a_j.gid, 1))$, the start round exists between the earliest and the latest in the first rounds of the additional cycles of good agents in a group candidate of a_j. Hence, in **Case (1)** (resp. **Case (2)**), by Observation 5, when a_i starts cycle c_i^*, a_j starts $\mathtt{REN}(\text{extendId}(a_j.\mathit{gid}, 1))$ (resp. cycle c_j^*) or a cycle of the AGREEID or MAKEGROUP stage between rounds $c_i^*[1] - t_{EX}$ and $c_i^*[t_{EX}]$. Furthermore, at the start of cycle c_j^* or $\mathtt{REN}(\text{extendId}(a_j.\mathit{gid}, 1))$, a_j has started the MAKEGROUP stage. In other words, a_j has finished at least one cycle of the AGREEID. Hence, at this time, the number of updates of $a_j.\mathit{lenCycle}$ is at least one less than the number of updates of $a_i.\mathit{lenCycle}$; thus, $|c_j^*| \le (1/2) \cdot |c_i^*|$ and $\mathit{lenCycle}_{rg} \le (1/2) \cdot |c_i^*|$ hold.

Here, when a_j does not start $\mathtt{REN}(\text{extendId}(a_j.\mathit{gid}, 1))$ or cycle c_j^* between rounds $c_i^*[1] - t_{EX}$ and $c_i^*[t_{EX}]$, let c_j^γ be a cycle of the AGREEID or MAKEGROUP stage that a_j starts during this period. By the behavior of `MakeReliableGroup` and Line 4 of Algorithm 12, a_j executes $\mathtt{REN}(\text{extendId}(a_j.\mathit{id}, 0))$ or $\mathtt{REN}(\text{extendId}(a_j.\mathit{guidepostId}, 0))$ without interruption for at least $(1/2) \cdot |c_j^\gamma| - t_{EX}$ rounds from round $c_j^*[(1/2) \cdot |c_j^*| + 1]$ at the latest. Cycle c_i^* completely includes the core periods of cycle c_j^* and the execution period of $\mathtt{REN}(\text{extendId}(a_j.\mathit{gid}, 1))$; hence, cycle c_i^* completely includes a core period of cycle c_j^γ. Thus, $|c_j^\gamma| \le (1/2) \cdot |c_i^*|$ holds.

We now prove this lemma in **Case (1)**. First, we consider the following case: when a_i starts cycle c_i^*, a_j starts $\mathtt{REN}(\text{extendId}(a_j.\mathit{gid}, 1))$ between rounds $c_i^*[1] - t_{EX}$ and $c_i^*[t_{EX}]$. It holds that $\mathit{lenCycle}_{rg} \le (1/2) \cdot |c_i^*|$; therefore, a_j transitions into a terminal state by round $r_{min} + \mathit{lenCycle}_{rg} \le c_i^*[t_{EX}] + \mathit{lenCycle}_{rg} \le c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$ by Corollary 6. Next, we consider the following case: when a_i starts cycle c_i^*, a_j starts cycle c_j^γ between rounds $c_i^*[1] - t_{EX}$ and $c_i^*[t_{EX}]$. Because $\mathit{lenCycle}_{rg} = |c_j^\gamma| \le (1/2) \cdot |c_i^*|$ holds, and an agent extends the length of its cycle by doubling that length, $(1/2) \cdot |c_i^*| \equiv 0 \pmod{\mathit{lenCycle}_{rg}}$ holds. If $r_{ini} \le c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$ holds, the first half of cycle c_i^* completely includes the core period of the execution period of $\mathtt{REN}(\text{extendId}(a_j.\mathit{gid}, 1))$. Thus, a_j transitions into a terminal state by round $r_{min} + \mathit{lenCycle}_{rg} \le c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$. If $r_{ini} > c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$ holds, the first half of cycle c_i^* completely includes the core period of cycle c_j^γ, and the second half of cycle c_i^* completely includes the core period of the execution period of $\mathtt{REN}(\text{extendId}(a_j.\mathit{gid}, 1))$. Because $|c_j^\gamma| \ge 12 \cdot$

$(t_{REN}(\text{extendId}(a_j.id, 0))+1)$ holds, and t_{EX} is smaller than $t_{REN}(\text{extendId}(a_j.id, 0))$, a_j executes $\texttt{REN}(\text{extendId}(a_j.id, 0))$ or $\texttt{REN}(\text{extendId}(a_j.guidepostId, 0))$ without interruption for at least $(1/2) \cdot |c_j^{\gamma}| - t_{EX} \geq 6 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1) - 2t_{EX} > 4 \cdot (t_{REN}(\text{extendId}(a_j.id, 0))+1)$ rounds from round $c_i^*[1]$ to round $c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$.

We next prove this lemma in **Case (2)**. First, we consider the following case: $r_{ini} \leq c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$. By Lemma 35, a_j transitions into a terminal state by round $\max(r_{min} + lenCycle_{rg}, c_j^*[last - t_{EX}])$. From the discussion of **Case (1)**, round $r_{min} + lenCycle_{rg}$ exists by round $c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$. Because $|c_j^*| \leq (1/2) \cdot |c_i^*|$ holds, and an agent extends its cycle length by doubling that length, $(1/2) \cdot |c_i^*| \equiv 0 \pmod{|c_j^*|}$ holds. Thus, the first half of cycle c_i^* completely includes the core period of cycle c_j^*. Therefore, $c_j^*[last] \leq c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$ holds. Hence, a_j transitions into a terminal state by round $c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$. Next, we consider the following case: $r_{ini} > c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$. In this case, the first half of cycle c_i^* completely includes a core period of cycle c_j^{γ}. Because $|c_j^{\gamma}| \geq 12 \cdot (t_{REN}(\text{extendId}(a_j.id, 0))+1)$ holds, and t_{EX} is smaller than $t_{REN}(\text{extendId}(a_j.id, 0))$, a_j executes $\texttt{REN}(\text{extendId}(a_j.id, 0))$ or $\texttt{REN}(\text{extendId}(a_j.guidepostId, 0))$ without interruption for at least $(1/2) \cdot |c_j^{\gamma}| - 2t_{EX} > 4 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ rounds from round $c_i^*[1]$ to round $c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$.

Finally, we prove this lemma in **Case (3)**. First, we consider the following case: $r_{ini} \leq c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$. By Lemma 35 and the discussion of **Case (1)**, a_j transitions into a terminal state by round $r_{min} + lenCycle_{rg} \leq c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$. Next, we consider the case $r_{ini} > c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$. If $|c_i^*| = |c_j^*|$ holds, a_j executes $\texttt{REN}(\text{extendId}(a_j.id, 0))$ without interruption for at least $(1/2) \cdot |c_j^*| - 2t_{EX} \geq 6 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1) - 2t_{EX} > 4 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ rounds from round $c_i^*[1]$ to round $c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$ because t_{EX} is smaller than $t_{REN}(\text{extendId}(a_j.id, 0))$; otherwise, let c_j^{ε} be the cycle of a_j that includes round $c_i^*[t_{EX}]$. From the same discussion in **Case (2)**, the first half of cycle c_i^* completely includes the core period of cycle c_j^{ε}. Because $|c_j^{\gamma}| \geq 12 \cdot (t_{REN}(\text{extendId}(a_j.id, 0))+1)$ holds, and t_{EX} is smaller than $t_{REN}(\text{extendId}(a_j.id, 0))$, a_j executes $\texttt{REN}(\text{extendId}(a_j.id, 0))$ or $\texttt{REN}(\text{extendId}(a_j.guidepostId, 0))$ without interruption for at least $(1/2) \cdot |c_j^{\gamma}| - 2t_{EX} > 4 \cdot (t_{REN}(\text{extendId}(a_j.id, 0)) + 1)$ rounds from round $c_i^*[1]$ to round $c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$. □

In the following lemma, we prove that every good agent in A_{ec} meets all good agents, and all good agents in A_{ec} gather together with RG_{min} and transition into a terminal state.

Lemma **37.** *Let a_i be an agent in A_{ec} and c_i^γ be the first cycle of the* MakeCandidate *stage of a_i. The following propositions hold for a_i: (1) agent a_i meets all good agents during cycle $c_i^{\gamma-1}$; and (2) agent a_i transitions into a terminal state together with RG_{min} before round $c_i^\gamma[last]$.*

Proof. By the behavior of `MakeReliableGroup`, a_i executes `REN`(extendId($a_i.id, 0$)) without interruption throughout cycle $c_i^{\gamma-1}$. Agent a_i also satisfies Line 8 of Algorithm 8 at the beginning of cycle $c_i^{\gamma-1}$; thus, $|c_i^{\gamma-1}| \geq 6 \cdot (t_{REN}(\text{extendId}(a_i.id, 0)) + 1)$ holds.

We prove this lemma by induction on the order in which the agents in A_{ec} start the MakeCandidate stage. Let a_{ini} be the first agent in A_{ec} that starts the MakeCandidate stage. First, we prove this lemma for a_{ini} as the base case of our induction. Let c_{ini}^ε be the first cycle of the MakeCandidate stage of a_{ini}. To prove Proposition (1), we break it down into the following cases: **Case (1)** Agent a_{ini} meets a good agent a_j in $A_{ec} \setminus \{a_{ini}\}$ during cycle $c_{ini}^{\varepsilon-1}$; and **Case (2)** Agent a_{ini} meets a good agent a_j either in A_{fc} or in a reliable group of SRG_{ini}.

We first prove **Case (1)**. By Observation 2, a_{ini} and a_j start cycles $c_{ini}^{\varepsilon-1}$ and $c_j^{\varepsilon-1}$ almost simultaneously, and $|c_{ini}^{\varepsilon-1}| = |c_j^{\varepsilon-1}|$ holds. Because t_{EX} is smaller than $t_{REN}(\text{extendId}(a_{ini}.id, , 0))$, a_j stays during the core period of cycle $c_{ini}^{\varepsilon-1}$ or executes `REN`(extendId($a_j.id, 0$)) for at least $5 \cdot (t_{REN}(\text{extendId}(a_{ini}.id, 0)) + 1)$ rounds. By Lemma 15, a_{ini} meets a_j before $c_{ini}^{\varepsilon-1}[last]$.

Next, we prove **Case (2)**. Cycle $c_{ini}^{\varepsilon-1}$ is cycle c_{ini}^* or a later cycle. First, we consider the case where cycle $c_{ini}^{\varepsilon-1}$ is a cycle after cycle c_{ini}^*. By Lemma 35, a_j transitions into a terminal state by round r^*. Thus, a_j stays from round $c_{ini}^{\varepsilon-1}[t_{EX}]$ at the latest. By Lemma 15, a_{ini} meets a_j before $c_{ini}^{\varepsilon-1}[last]$. Next, we consider the case where cycle $c_{ini}^{\varepsilon-1}$ is cycle c_{ini}^*. By Lemma 36, if $r_{ini} \leq c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$ holds, a_j transitions into a terminal state by round $c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$, that is, it stays from round $c_i^*[(1/2) \cdot |c_i^*| + t_{EX}]$. By Lemma 15, a_{ini} meets a_j before $c_{ini}^{\varepsilon-1}[last]$. If $r_{ini} > c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$ holds, a_j executes `REN`(extendId($a_j.id, 0$)) or `REN`(extendId($a_j.guidepostId, 0$)) for at least $t_{REN}(\text{extendId}(a_j.id, 0)$ rounds without interruption from rounds $c_i^*[1]$ to $c_i^*[(1/2) \cdot |c_i^*| - t_{EX}]$. Because $|c_i^{\gamma-1}| \geq 6 \cdot (t_{REN}(\text{extendId}(a_i.id, 0)) + 1)$ holds, by Lemma 29, a_{ini} meets a_j before $c_{ini}^{\varepsilon-1}[last]$.

The above discussion shows that a_{ini} knows the IDs of all good agents at the beginning of cycle c^{ε}_{ini}. Every good agent either in A_{fc} or in a reliable group of SRG_{ini} transitions into a terminal state together with RG_{min} by round $c^{\varepsilon}_{ini}[t_{\mathtt{EX}}]$. Therefore, by Lemma 2, a_{ini} visits all nodes within $t_{\mathtt{REN}}(\text{extendId}(a_{ini}.id, 0))$ rounds, and a_{ini} meets RG_{min} together with RG_{min} by $c^{\varepsilon}_{ini}[t_{\mathtt{EX}} + t_{\mathtt{REN}}(\text{extendId}(a_{ini}.id, 0))]$. Hence, by Observation 8, a_{ini} transitions into a terminal state together with RG_{min} by round $c^{\varepsilon}_{ini}[t_{\mathtt{EX}}+t_{\mathtt{REN}}(\text{extendId}(a_{ini}.id, 0))]$.

Next, let A'_{ec} be a set of good agents in A_{ec} that start the MakeCandidate stage by round $c^{\gamma-1}_{i}[t_{\mathtt{EX}}]$. We assume that all agents in A'_{ec} transition into terminal states together with RG_{min} by round $c^{\gamma-1}_{i}[t_{\mathtt{EX}}+t_{\mathtt{REN}}(\text{extendId}(a_i.id, 0))]$. We prove this lemma for a_i. From the same discussion as a_{ini}, a_i meets all good agents in $A_{ec} \setminus A'_{ec}$ and all good agents either in A_{fc} or in a reliable group of SRG_{ini} by $c^{\gamma-1}_{i}[last]$. Thus, we consider the meeting of a_i and good agents in A'_{ec}. Time $t_{\mathtt{EX}}$ is smaller than $t_{\mathtt{REN}}(\text{extendId}(a_i.id, 0))$, and $a_i.id$ is larger than IDs of agents in A'_{ec}; hence, a_i executes $\mathtt{REN}(\text{extendId}(a_i.id, 0))$ for at least $4 \cdot (t_{\mathtt{REN}}(\text{extendId}(a_i.id, 0)) + 1)$ rounds from round $c^{\gamma-1}_{i}[t_{\mathtt{EX}} + t_{\mathtt{REN}}(\text{extendId}(a_i.id, 0)) + 1]$. By Lemma 15, a_{ini} meets all good agents in A'_{ec} by round $c^{\gamma-1}_{i}[last]$. From the same discussion as a_{ini}, a_i transitions into a terminal state together with RG_{min} by round $c^{\gamma}_{i}[t_{\mathtt{EX}} + t_{\mathtt{REN}}(\text{extendId}(a_i.id, 0))]$. □

Finally, we prove the correctness and the complexity of `ByzantineGathering`.

Theorem **4.** *Let n be the number of nodes, k be the number of agents, f be the number of weakly Byzantine agents, $X(n)$ be the number of rounds required to explore any network composed of n nodes, Λ_{good} be the length of the largest ID among good agents, and Λ_{all} be the length of the largest ID among agents. If the upper bound N of n is given to agents, and $k \geq 8f + 7$ holds, Algorithm 13 solves the gathering problem in $O(f \cdot \Lambda_{good} \cdot X(N))$ rounds using $O(k \cdot (\Lambda_{all} + \log(X(N))) + MS_{REN}(N, 2^{\Lambda_{good}}) + MS_{PCONS}(S)$ bits of agent memory.*

Proof. Let a_{max} be a good agent with the largest ID among good agents. By the behavior of `ByzantineGathering`, an agent executes `MakeReliableGroup` until the current node includes a reliable group. By Theorem 3, a reliable group is created in the $O(f \cdot t_{\mathtt{REN}}(\text{extendId}(a_{max}.id, 0)))$ rounds right after starting

MakeReliableGroup. Thus, round r_{ini} is in the $O(f \cdot t_{REN}(\text{extendId}(a_{max}.id, 0)))$ rounds right after starting MakeReliableGroup.

By Lemma 35 and the definition of r^*, all good agents in A_{fc} and any reliable group of $SRG_{ini} \setminus \{RG_{min}\}$ meet RG_{min} and transition into terminal states by round r^*. If A_{ec} is not empty, by Lemma 37, a good agent in A_{ec} meets RG_{min} and transitions into a terminal state together with RG_{min} by the last round of the first cycle of the MakeCandidate stage of the agent.

We then analyze the time complexity of ByzantineGathering. The above discussion reveals that the time required to achieve the gathering depends on whether A_{ec} is empty or not. All good agents start the MakeCandidate stage by the round that a_{max} operates as that stage for t_{EX} rounds. Thus, by the assumption of A_{ec}, if A_{ec} does not contain a_{max}, no good agents are included in A_{ec}. We consider two cases, that is, A_{ec} does not contain a_{max}, and A_{ec} contains a_{max}. Let a_i be a good agent and $lenCycle_{max}$ be the value of a_{max} in round $r_{ini} + t_{EX}$.

In the former case, by Lemma 35, all good agents in A_{fc} and any reliable group of $SRG_{ini} \setminus \{RG_{min}\}$ transition into terminal states by $\max(r_{min} + lenCycle_{rg}, c_i^*[last - t_{EX}])$. Because $r_{min} \le r_{ini} + t_{EX}$ and $lenCycle_{rg} \le lenCycle_{max}$ hold, round $r_{min} + lenCycle_{rg}$ exists by round $r_{ini} + t_{EX} + lenCycle_{max}$. Moreover, because a_i starts cycle c_i^* by round $r_{ini} + t_{EX}$, and $|c_i^*| \le lenCycle_{max}$ holds, round $c_i^*[last - t_{EX}]$ exists by round $r_{ini} + t_{EX} + lenCycle_{max}$. It holds that $lenCycle_{max} < 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ by Lemma 18; hence, $r_{ini} + t_{EX} + lenCycle_{max} \le r_{ini} + 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) + t_{EX}$ holds. All good agents achieve the gathering in $O(f \cdot t_{REN}(\text{extendId}(a_{max}.id, 0))) + t_{EX} + 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) = O(f \cdot t_{REN}(\text{extendId}(a_{max}.id, 0)))$ rounds.

In the latter case, r^* exists between rounds $c_{max}^*[last]$ and $c_{max}^*[last] + t_{EX}$ by Observation 9. Agent a_{max} also finishes the CollectID stage in the $O(t_{REN}(\text{extendId}(a_{max}.id, 0)))$ rounds right after starting MakeReliableGroup by Lemma 19. It holds that $lenCycle_{max} < 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$; therefore, all good agents achieve the gathering in $O(t_{REN}(\text{extendId}(a_{max}.id, 0))) + t_{EX} + 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) = O(t_{REN}(\text{extendId}(a_{max}.id, 0)))$ rounds.

Time $t_{REN}(\text{extendId}(a_{max}.id, 0))$ is $O(X(N) \log(|2 \cdot (a_{max}.id) + 1|)) = O(\Lambda_{good} \cdot X(N))$. Hence, all good agents gather at the same node and transition into terminal states in the $O(f \cdot \Lambda_{good} \cdot X(N))$ rounds right after starting `ByzantineGathering`.

Finally, we analyze the space complexity required for an agent a_i to execute `ByzantineGathering`. We first consider the amount of memory space required for a_i to keep every variable.

Case Variables *lenCycle*, *numRound*, and *numRemainRound*: Lemma 18 gives $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ as the upper bound on *lenCycle*. The upper bounds of *numRound* and *numRemainRound* are also equal to and less than that on *lenCycle*. Thus, the amounts of memory space of these variables are $O(\log(\Lambda_{good} \cdot X(N)))$ bits.

Case Variables *stage*, *ready*, and *endMakeCandidate*: Agent a_i stores a constant number of parameters to $a_i.stage$, $a_i.ready$, and $a_i.endMakeCandidate$; thus, the amounts of memory space of these variables are $O(1)$ bits.

Case Variables R, S_p, P_p, S_c, P_c, BL, S_{gid}, and S_{rg}: For R, S_p, and P_p, a_i stores only IDs of agents it has met; therefore, it stores at most k agent IDs to $a_i.R$, $a_i.S_p$, $a_i.P_p$, $a_i.S_{gid}$, and $a_i.S_{rg}$. By Lemma 25, `PCONS`$(a_i.S_p)$ and `PCONS`$(a_i.P_p)$ satisfy the PBC property; thus, a_i also stores at most k agent IDs to $a_i.S_c$ and $a_i.P_c$. By Lemma 28, an agent stores only IDs of Byzantine agents; therefore, a_i stores at most f agent IDs to $a_i.BL$. The amounts of memory space of these variables are $O(k \cdot \Lambda_{all})$ bits.

Case Variable *numCycle*: Agent a_i spends $O(f)$ cycles in the AgreeID and MakeGroup stages by Lemmas 25 and 26; therefore, the amount of memory space of the variable is $O(\log(f))$ bits.

Case Variable D: Agent a_i stores only IDs and the remaining rounds of agents at the same node to $a_i.D$; therefore, a_i stores at most k tuples to $a_i.D$. The amount of memory space of the variable is $O(k(\Lambda_{all} + \log(\Lambda_{good} \cdot X(N)))) = O(k(\Lambda_{all} + \log(X(N))))$ bits.

Case Variables *guidepostId* and *gid*: Agent a_i stores one ID on $a_i.guidepostId$ and $a_i.gid$. The upper bounds on these variables are the largest ID among good agents. Thus, the amounts of memory space of these variables are $O(\log(\Lambda_{good}))$ bits.

The amount of memory space required for a_i to keep every variable is $O(k(\Lambda_{all} + \log(X(N))))$ bits.

We next consider the amount of memory space required for a_i to execute the building blocks of the algorithm. As mentioned in Section 1, the amount of memory space of the exploration procedure is $O(\log N)$. When a_i executes `REN`, a_i gives at most $a_i.id$ as an input; hence, the amount of memory space of the rendezvous procedure is $MS_{REN}(N, 2^{\Lambda_{good}})$. When a_i executes `PCONS`, a_i gives a set S of agent IDs; therefore, the amount of memory space of the parallel Byzantine consensus algorithm is $MS_{PCONS}(S)$. The amount of memory space required for a_i to execute the building blocks of the algorithm is $O(\log N) + MS_{REN}(N, 2^{\Lambda_{good}}) + MS_{PCONS}(S)$.

The above discussions demonstrate that the space complexity required for an agent to execute `ByzantineGathering` is $O(k(\Lambda_{all} + \log(X(N)))) + MS_{REN}(N, 2^{\Lambda_{good}}) + MS_{PCONS}(S)$. □

3. Byzantine Gathering Algorithm with Simultaneous Termination

This section introduces an algorithm that achieves the gathering with simultaneous termination. We do not need any new assumptions to achieve this. We refer to the algorithm in the previous section as the previous algorithm. The previous algorithm makes all good agents meet at a single node, but does not make them transition into terminal states at the same time. Thus, the algorithm in this section aims to make all good agents transition into the terminal states at the same time by modifying the previous algorithm.

We will first provide an overview of the proposed algorithm. This algorithm causes good agents to delay their transition into terminal states until round r_g. Round r_g is when all good agents have started the MakeCandidate stage and met the reliable group with the smallest group ID, say RG_{min}. At this point, all good agents transition into terminal states at the same time. To do this, every good agent in RG_{min} calculates the upper bound r_g^* of time r_g using a common ID set, waits until r_g^*, and transitions into a terminal state at the end of round r_g^*. Letting a_i be a good agent that finishes the CollectID stage and exists together with RG_{min}, a_i follows the behavior of RG_{min}. When agents in RG_{min} stay at the current node (resp. transition into terminal states), a_i also stays at the current node (resp. transitions into a terminal state). This algorithm guarantees that round r_g^* is a round after all good agents finish the CollectID stage and meet RG_{min}.

Next, we will present details of the algorithm. A good agent a_i executes the previous algorithm, except for transitioning into a terminal state. Let r_{fr} be the round in which good agents in RG_{min} finish the `REN` execution with their group IDs. If a_i belongs to RG_{min}, that is, $a_i.S_{gid} \neq \emptyset \wedge a_i.gid \neq \infty \wedge a_i.gid = a_i.minGID$ holds, a_i calculates the upper bound r_g^* using $\max(a_i.S_c)$ in round r_{fr}. More concretely, a_i treats round $r_{fr} + 1 + t_{\mathtt{EX}} + 96 \cdot (t_{\mathtt{REN}}(\text{extendId}(\max(a_i.S_c), 0)) + 1)$ as round r_g^*. Subsequently, a_i stays until round r_g^* and transitions into a terminal state in round r_g^*. If a_i does not belong to RG_{min} and finishes the CollectID stage, that is, $a_i.stage \in \{MakeCandidate, \text{AgreeID}, MakeGroup\} \wedge a_i.S_{gid} \neq \emptyset \wedge a_i.gid > a_i.minGID$ holds, a_i updates S_{rg} and executes `FOLLOW`$(a_i.S_{rg})$ in each round after meeting a reliable group with a smaller group ID.

Theorem **5.** *Let n be the number of nodes, k be the number of agents, f be the number of weakly Byzantine agents, $X(n)$ be the number of rounds required to explore any network composed of n nodes, Λ_{good} be the length of the largest ID among good agents, and Λ_{all} be the length of the largest ID among agents. If the upper bound N of n is given to agents, and $k \geq 8f+7$ holds, the proposed algorithm solves the gathering problem with simultaneous termination in $O(f \cdot \Lambda_{all} \cdot X(N))$ rounds using $O(k \cdot (\Lambda_{all} + \log(X(N))) + MS_{REN}(N, 2^{\Lambda_{good}}) + MS_{PCONS}(S)$ bits of agent memory.*

Proof. We prove that all good agents gather at the same node before any good agent transitions into a terminal state. They then transition into terminal states at the same time. Let a'_{max} be an agent with the largest ID among agents. We consider two cases here.

First, we consider the reliable group RG_{min} with the smallest group ID. Let a_i be a good agent in RG_{min} and r_{pre} be the round in which a_i transitions into a terminal state in the previous algorithm. By Corollary 6, all good agents in RG_{min} finish the REN execution with their group ID at the same time at the same node in round r_{pre}. By Lemma 25 and Corollary 5, $a_i.S_c$ contains IDs of all good agents, and all good agents in RG_{min} have the same S_c; hence, $\max(a_i.S_c) \leq a'_{max}.id$ holds, and $\max(a_i.S_c)$ is the same as that of another good agent in RG_{min}. All good agents in RG_{min} output the same value as r_g^*. Hence, all good agents in RG_{min} stay at the same node until round r_g^* and transition into terminal states at the same time at the same node in round r_g^*.

Next, we consider another good agent a_j. Let a_{max} be an agent with the largest ID among good agents. If a_j finishes the CollectID stage before the first reliable group executes a rendezvous procedure with its group ID for t_{EX} rounds, a_j meets RG_{min} within $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1)$ rounds from round $r_{pre} + 1$ by Lemmas 18 and 35; otherwise, a_j meets RG_{min} within $96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) - T_{ini}$ rounds right after starting `MakeReliableGroup` by Lemmas 19 and 37. Consequently, $\max(r_{pre} + t_{EX} + 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) + 1, 96 \cdot (t_{REN}(\text{extendId}(a_{max}.id, 0)) + 1) - T_{ini}) \leq \max(r_{pre} + t_{EX} + 96 \cdot (t_{REN}(\text{extendId}(\max(a_i.S_c), 0)) + 1) + 1, 96 \cdot (t_{REN}(\text{extendId}(\max(a_i.S_c), 0)) + 1) - T_{ini}) \leq r_g^*$ holds since $\max(a_i.S_c) \geq a_{max}.id$ holds. Therefore, all good agents meet RG_{min} before round r_g^*. Agent a_j also knows the IDs of all good agents when a_j meets RG_{min} after finishing the CollectID stage by Corollary 1 and Lemma 19. By Observation 8, a_j recognizes RG_{min} as a reliable group at this time. Thus, a_j follows the behavior of RG_{min} after meeting RG_{min}. Correspondingly, a_j transitions into a terminal state at the same time as a_i together with RG_{min}.

Based on the discussion so far, all good agents gather at the same node before any good agent transitions into a terminal state. They then transition into terminal states at the same time.

Next, we analyze the time complexity of the proposed algorithm. By Theorem 4, round r_{pre} is in $O(f \cdot \Lambda_{good} \cdot X(n))$ rounds right after starting the previous algorithm, where Λ_{good} is the length of the largest ID among good agents. Agent a_i then waits for $t_{EX} + 96 \cdot (t_{REN}(\text{extendId}(\max(a_i.S_c), 0) + 1)$ rounds from round $r_{pre} + 1$. Since $\max(a_i.S_c) \leq a'_{max}.id$, $O(f \cdot \Lambda_{good} \cdot X(n)) + t_{EX} + 96 \cdot (t_{REN}(\text{extendId}(\max(a_i.S_c), 0) + 1) + 1 < O(f \cdot \Lambda_{good} \cdot X(n)) + t_{EX} + 96 \cdot (t_{REN}(\text{extendId}(a'_{max}.id, 0) + 1) + 1 = O(f \cdot \Lambda_{good} \cdot X(n)) + O(f \cdot t_{REN}(\text{extendId}(a'_{max}.id, 0)))$ holds. Time $t_{REN}(\text{extendId}(a'_{max}.id, 0))$ is $O(X(N) \log(|2 \cdot (a'_{max}.id) + 1|)) = O(\Lambda_{all} \cdot X(N))$. Hence, all good agents gather at the same node and transition into the terminal states at the same time in $O(f \cdot \Lambda_{good} \cdot X(n)) + O(f \cdot t_{REN}(\text{extendId}(a'_{max}.id, 0))) = O(f \cdot \Lambda_{all} \cdot X(n))$ rounds right after the first good agent wakes up.

Finally, we analyze the space complexity required for an agent a_i to execute the proposed algorithm. The proposed algorithm uses all variables and building blocks of the previous algorithm; hence, the space complexity is at least $O(k(\Lambda_{all} + \log(X(N)))) + MS_{REN}(N, 2^{\Lambda_{good}}) + MS_{PCONS}(S)$ bits by Theorem 4. In addition, a_i uses a variable var_{ubr} to keep round r_g^* and a variable var_{rr} to hold the remaining rounds until round r_g^* in the proposed algorithm; therefore, we analyze the amount of the memory space of var_{ubr} and var_{rr} in the rest of this paragraph. Let a_ℓ be the agent with the largest ID in $a_i.S_c$. Agent a_i stores the upper bound on the time required for a_ℓ to finish the CollectID stage to var_{ubr}. The upper bound on the time is also derived from $O(\log(a_\ell.id) \cdot X(N))$ by Lemma 18. Thus, the amount of memory space of var_{ubr} is $O(\log(\Lambda_{all} \cdot X(N)))$ bits. The upper bound on var_{rr} is equal to the upper bound on var_{ubr}; therefore, the amount of the memory space of var_{rr} is also $O(\log(\Lambda_{all} \cdot X(N)))$ bits. The space complexity required for a_i to execute the proposed algorithm is $O(k \cdot (\Lambda_{all} + \log(X(N)))) + MS_{REN}(N, 2^{\Lambda_{good}}) + MS_{PCONS}(S) + O(\log(\Lambda_{all} \cdot X(N))) = O(k \cdot (\Lambda_{all} + \log(X(N)))) + MS_{REN}(N, 2^{\Lambda_{good}}) + MS_{PCONS}(S)$ bits. □

4. Summary

In this part, we provided two gathering algorithms with different termination characteristics in the presence of $O(k)$ Byzantine agents. These algorithms have a low time complexity and require a small number of good agents. More specifically, if N is given to agents, and at least $8f + 7$ agents exist in the network, the first algorithm achieves the gathering with non-simultaneous termination in $O(f \cdot \Lambda_{good} \cdot X(N))$ rounds, and the second one achieves the gathering with simultaneous termination in $O(f \cdot \Lambda_{all} \cdot X(N))$ rounds. In these algorithms, similarly to agents in Part IV, several good agents first create a reliable group, and then the reliable group collects the other good agents; as a result, all good agents gather at a single node. To create a reliable group, several good agents make a common ID set by simulating a parallel Byzantine consensus algorithm and realize gathering by using the common ID set.

Part VI

Discussion

In this section, we examine the time improvements and extensions to the proposed algorithms. For extensions, we explore the possibility of concurrent execution with the existing algorithm [11] and the proposed algorithms, and the solvability in dynamic networks.

First, we mention improvements in the time complexities of the proposed algorithms. Both proposed algorithms include the factor of the length Λ_{all} of the largest ID among agents in their time complexity; we investigate whether this can be changed to the length Λ_{good} of the largest ID among good agents, similar to the existing algorithm [11]. To state the conclusion upfront, this is challenging because the estimation of the termination time for these algorithms is influenced by the presence of Byzantine agents. In the existing algorithm, an agent a_i starts the rendezvous algorithm $\texttt{REN}(a_i.id)$. Whenever the members of the agents acting together with a_i change, a_i stops the current execution of REN and starts $\texttt{REN}(\ell_{min})$ using the smallest ID ℓ_{min} among IDs of agents that are considered trustworthy at the current node. For a sufficiently long time T calculated from n and ℓ_{min}, when the agents execute $\texttt{REN}(\ell_{min})$ for T rounds without a change in the membership, agents can determine that all good agents gather at a single node. Whereas, in both proposed algorithms, agents make this determination based on the calculation of the expected arrival time of the agent with the largest ID among the collected IDs. Thus, in the estimation of the termination time, while the existing algorithm selects the smallest ID at the current node, the proposed algorithms choose the largest ID in the network. This difference implies that while the existing algorithm may select an ID that is not greater than the smallest ID among the good agents at the current node, the proposed algorithms may choose an ID of a Byzantine agent that is larger than the largest ID among good agents. Therefore, to change Λ_{all} in their time complexities to Λ_{good}, it is necessary to remove the factor of ID length from the execution time of the rendezvous algorithm, since the estimation of the termination time is based on the execution time. Alternatively, it is essential to design a termination method that does not depend on the largest ID among the collected IDs.

Next, we explore the possibility of concurrent execution with the existing algorithm [11] and the proposed algorithms. To achieve this, we note that the behaviors of agents in each algorithm can be composed of executing the exploration procedure EX and waiting for the execution time t_{EX} of EX. Therefore, we can divide the execution of each algorithm into t_{EX} rounds and execute each algorithm sequentially in every t_{EX} rounds interval. In the other words, let $Algo_0$ be the existing algorithm [11], $Algo_1$ be the algorithm proposed in Part IV, and $Algo_2$ be the algorithm proposed in Part V, we execute t_{EX} rounds of $Algo_1$, followed by t_{EX} rounds of $Algo_2$, and then t_{EX} rounds of $Algo_0$, in a continuous cycle. However, we identify two issues with this idea. The first issue is that the start node in the $j+1$-th $(j \geq 1)$ t_{EX} round of algorithm $Algo_i$ $(0 \leq i \leq 2)$ may differ from the end node in the j-th t_{EX} round of algorithm $Algo_i$. This issue leads to a problem where agents become separated from each other during algorithm operations that involve moving together with other agents. The second issue is that agents may start $Algo_i$ at different times because the start times for the algorithm among them may differ. Due to this issue, when an agent executes $Algo_i$ for t_{EX} rounds, we cannot guarantee that the other agents execute $Algo_i$ during the same period. It is not immediately apparent to address these issues, but the proposal of new mechanisms or approaches is essential. Such efforts would facilitate the efficient combination and use of different algorithms. We are confident that providing these solutions would significantly contribute to developing more adaptable and efficient agent-based systems.

Finally, we explore whether the proposed algorithms are solvable for execution in dynamic networks. The behaviors of agents in both algorithms are based on an exploration procedure that guarantees an agent alone visits every node in an arbitrary graph at least once, using the number of nodes as input, and waiting for the execution period of the exploration procedure, which can be calculated from the number of nodes. Therefore, if there exists a similar exploration suited for dynamic networks, we believe that the gathering in dynamic networks is solvable. Research on exploring algorithms in dynamic networks has been extensive in recent years [22], but, to the best of my knowledge, the exploration algorithm with the above properties has not yet been found.

Part VII

Conclusion

This dissertation focuses on the gathering problem in synchronous environments with Byzantine agents. This problem requires that all good agents, initially scattered through the network, meet at a single node and declare the termination at the same time. As an algorithm tolerates Byzantine agents, the fastest algorithm is one proposed by Dieudonné et al. [11]. This algorithm tolerates any number of Byzantine agents and works in $O(n^4 \cdot \Lambda_{good} \cdot X(n))$ rounds if the number n of nodes is given to agents, where Λ_{good} is the length of the largest ID among good agents, and $X(n)$ is the time required to visit all nodes of any n-nodes network; however, its time complexity is not insignificant.

We show two efficient algorithms that solve the gathering problems in synchronous environments with Byzantine agents, assuming that Byzantine agents constitute few numbers. In Part IV, we provided the gathering algorithm in $O((f+\Lambda_{all}) \cdot X(N))$ rounds if agents know the upper bound N on n and at least $(4f+4)(f+1)$ agents exist in the network, where Λ_{all} is the length of the largest ID among agents. This algorithm greatly reduces the time complexity compared to that [11]. In Part V, we proposed the gathering algorithm in $O(f \cdot \Lambda_{all}) \cdot X(N))$ rounds if agents know N and at least $8f+7$ agents exist in the network. This algorithm is faster than that [11] and requires a smaller number of good agents than that in Part IV. By proposing these algorithms, trade-offs between the ratio of non-Byzantine agents to Byzantine agents and the time complexity in gathering problems in the presence of Byzantine agents are indicated.

From now on, we show the future tasks of our work. Our algorithm requires at least $8f+7$ agents for the simulation of a Byzantine consensus algorithm to achieve the gathering; however, we should investigate whether it is possible to execute this simulation with fewer agents. Another future task is to investigate the space complexity required to achieve the gathering problem in the presence of Byzantine agents. While it is known that the minimum number of memory bits in the gathering problem without Byzantine agents is $\Theta(\log n)$ [6], algorithms for solving the gathering problem with Byzantine agents require $\Omega(k \cdot \Lambda_{all})$ memory bits; presenting a significant gap. We should study whether it is possible to solve the gathering problem in the presence of Byzantine agents using $o(k \cdot \Lambda_{all})$ memory bits.

Acknowledgements

I extend my deepest appreciation to everyone who has supported me. First and foremost, my heartfelt thanks go to Professor Michiko Inoue for her unparalleled guidance and support during my Master's and Doctoral studies. Her insightful guidance has laid the groundwork for the composition of my thesis writing and presentation skills. Secondly, my sincere appreciation is directed towards Professor Shoji Kasahara for his insightful feedback and invaluable suggestions. His expertise has greatly contributed to the refinement of my research. Heartfelt thanks are extended to Professor Fukuhito Ooshita, who belongs to Fukui University of Technology, for his dedicated mentorship and expert advice. He generously provided numerous comments derived from his vast knowledge and meticulous attention during my thesis writing, enabling me to compose a well-considered paper. I am indebted to Associate Professor Junya Nakamura, who belongs to Toyohashi University of Technology, for his warm encouragement and support. He provided various comments for my thesis and presentations plus substantial assistance with my writing environment, allowing me to concentrate on my research. My gratitude goes out to Associate Professor Yuichi Sudo, who belongs to Hosei University, for his financial support and research opportunities provided by him. His support afforded me financial flexibility, and the research opportunities he provided allowed me to gain a wealth of experience and insights. I am thankful to Assistant Professor Ryota Eguchi for his dedicated support and encouragement. He often visited me to engage in casual conversations, which helped me stay mentally balanced and dedicated to my research without succumbing to stress. I convey my sincere gratitude to all my Laboratory members for their support over the course of my five-year journey as a student. Lastly, my deepest gratitude is reserved for my family and the friends I made during my time at the National Institute of Technology, Ishikawa College, Toyohashi University of Technology, and Nara Institute of Science and Technology, whose support and care have been constants throughout my academic journey.

References

[1] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006.

[2] Sébastien Bouchard, Yoann Dieudonné, and Bertrand Ducourthial. Byzantine gathering in networks. *Distributed Computing*, 29(6):435–457, 2016.

[3] Sébastien Bouchard, Yoann Dieudonné, and Anissa Lamani. Byzantine gathering in polynomial time. *Distributed Computing*, 35(3):235–263, 2022.

[4] Sébastien Bouchard, Yoann Dieudonné, and Andrzej Pelc. Want to gather? no need to chatter! *SIAM Journal on Computing*, 52(2):358–411, 2023.

[5] Jérémie Chalopin, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29:187–205, 2016.

[6] Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Computing*, 25(2):165–178, 2012.

[7] Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. Time versus space trade-offs for rendezvous in trees. *Distributed Computing*, 27(2):95–109, 2014.

[8] Jurek Czyzowicz, Andrzej Pelc, and Arnaud Labourel. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms*, 8(4):37:1–37:14, 2012.

[9] Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

[10] Yoann Dieudonné and Andrzej Pelc. Anonymous meeting in networks. *Algorithmica*, 74(2):908–946, 2016.

[11] Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1–28, 2014.

[12] Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM Journal on Computing*, 44(3):844–867, 2015.

[13] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[14] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors. *Distributed Computing by Mobile Entities*, volume 683 of *Lecture Notes in Computer Science*. Springer Cham, 1 edition, 2019.

[15] Pierre Fraigniaud and Andrzej Pelc. Deterministic rendezvous in trees with little memory. In *Distributed Computing, 22nd International Symposium, DISC 2008*, volume 5218, pages 242–256. Springer, 2008.

[16] Pierre Fraigniaud and Andrzej Pelc. Delays induce an exponential memory gap for rendezvous in trees. *ACM Transactions on Algorithms*, 9(2):17:1–17:24, 2013.

[17] Pankaj Khanchandani and Roger Wattenhofer. Byzantine agreement with unknown participants and failures. In *35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021*, pages 952–961. IEEE, 2021.

[18] Michal Koucký. Universal traversal sequences with backtracking. *Journal of Computer and System Sciences*, 65(4):717–726, 2002.

[19] Dariusz R. Kowalski and Adam Malinowski. How to meet in anonymous network. *Theoretical Computer Science*, 399(1-2):141–156, 2008.

[20] Evangelos Kranakis, Danny Krizanc, Euripides Markou, Aris Pagourtzis, and Felipe Ramírez. Different speeds suffice for rendezvous of two agents on arbitrary graphs. In *43rd International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2017*, volume 10139, pages 79–90. Springer, 2017.

[21] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[22] Giuseppe Antonio Di Luna. Mobile agents on dynamic graphs. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 549–584. Springer, 2019.

[23] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006.

[24] Avery Miller and Andrzej Pelc. Time versus cost tradeoffs for deterministic rendezvous in networks. *Distributed Computing*, 29(1):51–64, 2016.

[25] Avery Miller and Ullash Saha. Fast byzantine gathering with visibility in graphs. In *Algorithms for Sensor Systems (ALGOSENSORS 2020)*, pages 140–153. Springer International Publishing, 2020.

[26] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. Fast deterministic gathering with detection on arbitrary graphs: The power of many robots. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2023*, pages 47–57. IEEE, 2023.

[27] Fukuhito Ooshita, Ajoy K Datta, and Toshimitsu Masuzawa. Self-stabilizing rendezvous of synchronous mobile agents in graphs. In *Stabilization, Safety, and Security of Distributed Systems: 19th International Symposium, SSS 2017*, pages 18–32. Springer, 2017.

[28] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[29] Andrzej Pelc. Deterministic gathering with crash faults. *Networks*, 72(2):182–199, 2018.

[30] Andrzej Pelc. Deterministic rendezvous algorithms. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 423–454. Springer, 2019.

[31] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):17:1–17:24, 2008.

[32] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms*, 10(3):12:1–12:15, 2014.

[33] Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue. Byzantine-tolerant gathering of mobile agents in arbitrary networks with authenticated whiteboards. *IEICE Transactions on Information and Systems*, 101-D(3):602–610, 2018.

[34] Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue. Byzantine-tolerant gathering of mobile agents in asynchronous arbitrary networks with authenticated whiteboards. *IEICE Transactions on Information and Systems*, 103-D(7):1672–1682, 2020.

Publication list

Peer-Reviewed Journal Papers

[1] Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue, Weakly Byzantine Gathering with a Strong Team, IEICE Transaction on Information and Systems, 105(3), 541-555, 2022.

[2] Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue, Fast gathering despite a linear number of weakly Byzantine agents, Concurrency and Computation: Practice and Experience, 2024. (Accepted)

Peer-Reviewed Conference Papers

[1] Jion Hirose, Masashi Tsuchida, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue, Brief Announcement : Gathering with a strong team in weakly Byzantine environments, 27th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2020), 2020.

[2] Jion Hirose, Junya Nakamura, Fukuhito Ooshita and Michiko Inoue, Gathering with a strong team in weakly Byzantine environments, 22nd International Conference on Distributed Computing and Networking (ICDCN 2021), 2021.

[3] Jion Hirose, Junya Nakamura, Fukuhito Ooshita and Michiko Inoue, Brief Announcement : Gathering despite a linear number of weakly Byzantine agents, 41st ACM Symposium on Principles of Distributed Computing (PODC 2022), 2022.

[4] Jion Hirose, Junya Nakamura, Fukuhito Ooshita and Michiko Inoue, Gathering despite a linear number of weakly Byzantine agents, 14th International Workshop on Parallel and Distributed Algorithms and Applications (PDAA 2022), 2022.