PAPER Special Section on Foundations of Computer Science - New Trends of Theory of Computation and Algorithm -

Weakly Byzantine Gathering with a Strong Team*

Jion HIROSE[†a)], ***Nonmember***, **Junya NAKAMURA**[††], **Fukuhito OOSHITA**[†], ***Members***, ***and*** **Michiko INOUE**[†], ***Fellow***

SUMMARY We study the gathering problem requiring a team of mobile agents to gather at a single node in arbitrary networks. The team consists of k agents with unique identifiers (IDs), and f of them are weakly Byzantine agents, which behave arbitrarily except falsifying their identifiers. The agents move in synchronous rounds and cannot leave any information on nodes. If the number of nodes n is given to agents, the existing fastest algorithm tolerates any number of weakly Byzantine agents and achieves gathering with simultaneous termination in $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$ rounds, where $|\Lambda_{good}|$ is the length of the maximum ID of non-Byzantine agents and $X(n)$ is the number of rounds required to explore any network composed of n nodes. In this paper, we ask the question of whether we can reduce the time complexity if we have a strong team, i.e., a team with a few Byzantine agents, because not so many agents are subject to faults in practice. We give a positive answer to this question by proposing two algorithms in the case where at least $4f^2 + 9f + 4$ agents exist. Both the algorithms assume that the upper bound N of n is given to agents. The first algorithm achieves gathering with non-simultaneous termination in $O((f + |\Lambda_{good}|) \cdot X(N))$ rounds. The second algorithm achieves gathering with simultaneous termination in $O((f + |\Lambda_{all}|) \cdot X(N))$ rounds, where $|\Lambda_{all}|$ is the length of the maximum ID of all agents. The second algorithm significantly reduces the time complexity compared to the existing one if n is given to agents and $|\Lambda_{all}| = O(|\Lambda_{good}|)$ holds.

key words: *distributed algorithm, deterministic algorithm, mobile agents, gathering, Byzantine faults*

1. Introduction

1.1 Background

Mobile agents (in short, agents) are software programs that move autonomously and perform various tasks in a distributed system. A problem of collecting multiple agents at the same node without previous agreement is called a *gathering*, and this problem has been widely studied from the theoretical aspect of distributed systems [1]. By solving this problem, the agents can exchange information with each other more efficiently and easily carry out future cooperative behaviors.

In operations of large-scale distributed systems, we cannot avoid facing faults of agents. Among them, Byzantine faults are known to be the worst faults because Byzantine faults do not make any assumption about the behavior of faulty agents (called *Byzantine agents*). For example, Byzantine agents can stop and move at any time apart from their algorithm, and tell arbitrary wrong information to other agents.

In this paper, we consider the gathering problem with Byzantine agents and propose two deterministic synchronous gathering algorithms for the problem.

1.2 Related Works

The gathering problem has been studied for the first time by Schelling [5]. In that paper, the author studied the gathering problem of exactly two agents, called the rendezvous problem. After that, the rendezvous problem and its generalization, the gathering problem, have been widely studied in various environments that combine the assumptions on agent synchronization, anonymity, presence/absence of memory on a node (called whiteboard), presence/absence of randomization, topology, etc. [1]. The purpose of these studies is to clarify the solvability of the gathering problem and its costs (e.g., time, the number of moves, memory space, etc.) if solvable. The rest of this section describes the existing results for the deterministic gathering problem in arbitrary networks, on which we focus in this paper.

Many of the papers dealing with the rendezvous problem assume that agents move synchronously in a network and that agents cannot leave any information on nodes, that is, whiteboards do not exist [1]. These works have studied the solvability of the rendezvous and, if solvable, the time required to solve the problem. If agents are anonymous (i.e., do not have IDs), the deterministic rendezvous cannot be achieved in some symmetric graphs because the symmetry cannot be broken. In the literature [6]–[9], rendezvous algorithms have been proposed in any graph by assuming a unique ID for each agent. Dessmark et al. [6] have proposed an algorithm to achieve the rendezvous in polynomial time of n, λ, and τ, where n is the number of nodes, λ is the smallest ID among agents, and τ is the difference between the startup times of agents. Kowalski et al. [7] and Ta-shma et al. [9] have improved the time complexity by proposing algorithms that achieve the rendezvous in time independent

Manuscript received March 26, 2021.
Manuscript revised July 21, 2021.
Manuscript publicized October 11, 2021.

[†]The authors are with Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.
[††]The author is with Toyohashi University of Technology, Toyohashi-shi, 441–8580 Japan.

*This work was partially supported by JSPS KAKENHI Grant Numbers 18K11167, 20H04140, and JP20KK0232, and JST SICORP Grant Number JPMJSC1806. A preliminary extended abstract of this paper appears in the proceedings of the 22nd International Conference on Distributed Computing and Networking.

a) E-mail: hirose.jion.hg4@is.naist.jp

DOI: 10.1587/transinf.2021FCP0011

Table 1 A summary of synchronous Byzantine gathering algorithms with unique IDs. Here, input is the information that an agent has at the start of an algorithm, n is the number of nodes, N is the upper bound of n, $|\lambda_{good}|$ is the length of the smallest ID among non-Byzantine agents, $|\Lambda_{good}|$ is the length of the largest ID among non-Byzantine agents, $|\Lambda_{all}|$ is the length of the largest ID among agents, k is the number of agents, f is the number of Byzantine agents, and F is the upper bound of f.

	Input	Byzantine	Condition of #Byzantine agents	Simultaneous termination	Time complexity
[2]	n	Weak	$f+1 \leq k$	Possible	$O(n^4 \cdot \lvert\Lambda_{good}\rvert \cdot X(n))$
[2]	F	Weak	$2F+2 \leq k$	Possible	Poly. of n & $\lvert\Lambda_{good}\rvert$
[2]	n, F	Strong	$3F+1 \leq k$	Possible	Exp. of n & $\lvert\Lambda_{good}\rvert$
[2]	F	Strong	$5F+2 \leq k$	Possible	Exp. of n & $\lvert\Lambda_{good}\rvert$
[3]	n, F	Strong	$2F+1 \leq k$	Possible	Exp. of n & $\lvert\Lambda_{good}\rvert$
[3]	F	Strong	$2F+2 \leq k$	Possible	Exp. of n & $\lvert\Lambda_{good}\rvert$
[4]	$\lceil \log \log n \rceil$	Strong	$5f^2+7f+2 \leq k$	Possible	Poly. of n & $\lvert\lambda_{good}\rvert$
Proposed Algorithm 1	N	Weak	$4f^2+9f+4 \leq k$	Impossible	$O((f+\lvert\Lambda_{good}\rvert) \cdot X(N))$
Proposed Algorithm 2	N	Weak	$4f^2+9f+4 \leq k$	Possible	$O((f+\lvert\Lambda_{all}\rvert) \cdot X(N))$

of τ. In addition, Millar et al. [8] have analyzed the trade-off between the time required for rendezvous and the number of moves. On the other hand, some papers [10]–[12] have investigated the memory space, the time, and the number of moves required to achieve the deterministic rendezvous without assuming a unique ID of each agent. Since the rendezvous cannot be accomplished for some initial arrangements of agents and graphs without unique IDs, they have proposed algorithms for limited graphs and initial arrangements. Fraigniaud et al. [11], [12] have proposed algorithms for trees, and Czyzowicz et al. [10] have proposed an algorithm for arbitrary graphs when initial arrangements of agents are not symmetric.

While many papers deal with the rendezvous problem in synchronous environments, some papers assume asynchronous environments where different agents move at different constant speeds or move asynchronously. In the latter case, speeds of agents in each time are always determined by the adversary. For more details, please refer to the literature [13]–[16] for a finite graph and the literature [17]–[19] for an infinite graph.

Recently some papers [2]–[4], [20], [21] have studied the gathering problem in the presence of Byzantine agents, which we also address in this study. Table 1 shows this research and the related researches that are closest to this research. These studies assume agents with unique IDs and consider two types of Byzantine agents depending on whether they can falsify their own IDs. *Weakly Byzantine agents* perform arbitrary behaviors except falsifying their own IDs, and *strongly Byzantine agents* perform arbitrary behaviors, including falsifying their own IDs.

Dieudonné et al. [2] have studied the gathering problem in synchronous environments where k agents exist in an n-node arbitrary network and at most F of them are Byzantine. They have proposed the two gathering algorithms for weakly Byzantine agents. The first gathering algorithm achieves the time complexity of $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$ if n is given to agents. The second gathering algorithm achieves the time complexity that is polynomial of n and $|\Lambda_{good}|$ if F is given to agents. Here, $|\Lambda_{good}|$ is the length of the largest ID among non-Byzantine agents, and $X(n)$ is the number of rounds required to explore any network composed of n nodes. The former algorithm and the latter algorithm work if at least one and at least $F+2$ non-Byzantine agents exist, respectively. These algorithms are optimal in terms of the number of tolerable Byzantine agents because they match the number of non-Byzantine agents required to solve the gathering problem under the conditions. For strongly Byzantine agents, two gathering algorithms have been proposed. The first algorithm achieves the gathering with $2F+1$ non-Byzantine agents under the condition that n and F are given to agents. The second algorithm achieves the gathering with $4F+2$ non-Byzantine agents under the condition that F is given to agents. On the other hand, the lower bounds of the numbers of non-Byzantine agents required to solve the gathering problem under the former condition and the latter condition are $F+1$ and $F+2$, respectively. Bouchard et al. [3] have proposed algorithms with the number of non-Byzantine agents that matches these lower bounds in the presence of strongly Byzantine agents. However, the time complexities of all the above algorithms for strongly Byzantine agents are exponential of n and $|\Lambda_{good}|$. Bouchard et al. [4] have proposed the gathering algorithm with polynomial time complexity for the first time in the presence of strongly Byzantine agents in synchronous environments. The gathering algorithm operates under the assumption that $\lceil \log \log n \rceil$ is given to agents and at least $5f^2+6f+2$ non-Byzantine agents exist in the network, where f is the number of Byzantine agents.

In synchronous environments with strongly Byzantine agents, Miller et al. [22] have reduced the time complexity of gathering by an additional assumption. They give an algorithm with the time complexity of $O(kn^2)$ for $k \geq 2f+1$ on the assumption that an agent can observe the subgraph induced by nodes within distance D_r from its current node and the states of agents in the subgraph, where f is the number of Byzantine agents and D_r is the radius of the graph.

Tsuchida et al. [20] have studied the gathering algorithm in synchronous environments with weakly Byzantine agents under the assumption that each node is equipped with an authenticated whiteboard, where each agent can leave information on its dedicated area but every agent can read all information. If the upper bound F of the number of weakly Byzantine agents is given to agents, the gathering

algorithm with the time complexity of $O(Fm)$ has been proposed, where m is the number of edges. Tsuchida et al. [21] have proposed the gathering algorithms in asynchronous environments in the presence of weakly Byzantine agents under the same assumption of authenticated whiteboards.

1.3 Our Contributions

We seek an algorithm that achieves the gathering with small time complexity in synchronous environments with weakly Byzantine agents. When agents cannot leave any information on nodes and cannot get information on nodes other than the current one, the existing fastest algorithm is the one proposed by Dieudonné et al. [2]. The algorithm tolerates any number of weakly Byzantine agents and achieves the gathering *with simultaneous termination*, and its time complexity is $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$, where n is the number of nodes, $|\Lambda_{good}|$ is the length of the largest ID among non-Byzantine agents, and $X(n)$ is the number of rounds required to explore any network composed of n nodes. Miller et al. [22] and Tsuchida et al. [21] have proposed faster algorithms with the assumptions of ability to observe distant nodes and authenticated whiteboards, respectively, but these assumptions are strong and greatly restrict the behavior of Byzantine agents.

In this paper, we reduce the time complexity by taking advantage of a *strong team*, that is, a team (collection of agents) with a few Byzantine agents. Since not so many agents are subject to faults in practice, the assumption of a strong team is reasonable. We propose two gathering algorithms that tolerate f weakly Byzantine agents in the case where a strong team composed of at least $4f^2 + 9f + 4$ agents exist (see Table 1). Both the algorithms take the upper bound N of n as input. The first algorithm achieves the gathering *with non-simultaneous termination* and its time complexity is $O((f + |\Lambda_{good}|) \cdot X(N))$, where $|\Lambda_{good}|$ is the length of the maximum ID of non-Byzantine agents. The second algorithm achieves the gathering *with simultaneous termination* and its time complexity is $O((f + |\Lambda_{all}|) \cdot X(N))$, where $|\Lambda_{all}|$ is the length of the maximum ID of all agents. If n is given to agents, the second algorithm significantly reduces the time complexity compared to that of Dieudonné et al. in case of $|\Lambda_{all}| = O(|\Lambda_{good}|)$.

2. Preliminaries

(1) Distributed systems

A distributed system is modeled by a connected undirected graph $G = (V, E)$, where V is a set of n nodes, and E is a set of edges. If an edge $\{u, v\} \in E$ exists between the nodes $u, v \in V$, u and v are said to be adjacent. A set of adjacent nodes of node v is denoted by $N_v = \{u \mid \{v, u\} \in E\}$. The degree of node v is defined as $d(v) = |N_v|$. Each edge incident to node v is locally and uniquely labeled by function $P_v : \{\{v, u\} \mid u \in N_v\} \to \{1, 2, \ldots, d(v)\}$ that satisfies $P_v(\{v, u\}) \neq P_v(\{v, w\})$ for edges $\{v, u\}$ and $\{v, w\}$ $(u \neq w)$. $P_v(\{v, u\})$ is called the port number of an edge $\{v, u\}$ on node v. Nodes have neither ID nor memory. Time is discretized, and each discretized time is called a round.

(2) Mobile agents

There are k agents $a_1, a_2, \ldots, a_k$ in the system. Agents cannot mark visited nodes or traversed edges in any way. Each agent a_i has a unique ID denoted by $a_i.ID \in \mathbb{N}$, but does not know a priori the IDs of other agents. Also, agents know the upper bound N of the number of nodes, but they do not know k, the topology of the graph, or n. The amount of agent memory is unlimited, and the contents of memory are not changed during a move through an edge.

The adversary wakes up at least one agent at the first round. We call an agent that did not start at the first round dormant. A dormant agent is woken up when the adversary wakes up the agent at some round or an agent visits the starting node of the dormant agent. Note that the adversary can awake dormant agents at different rounds.

An agent is modeled as a state machine (S, δ). Here, S is a set of agent states, and a state is represented by a tuple of the values of all the variables that an agent has. The state transition function δ outputs the next agent state, whether the agent stays or leaves, and the outgoing port number if the agent leaves. The outputs are determined from the current agent state, the states of other agents at the same node, the degree of the current node, and the entry port. An agent has a special state representing the termination of an algorithm. After reaching the state, the agent never executes the algorithm. If several agents are at node v, the agents can read all the information that they have (even if some of them have terminated). However, if two agents traverse the same edge simultaneously in different directions, the agents do not notice this fact. When an agent enters a node v via an edge $\{u, v\}$, it learns the degree $d(v)$ of v and the port number $P_v(\{v, u\})$. Agents execute the algorithm synchronously. That is, at the beginning of a round, each agent reads states of all agents at the current node, executes the state transition. If an agent decides to move, it arrives at the destination node before the beginning of the next round. Note that, in each round, all agents at a single node obtain the same information of states of the agents.

(3) Byzantine agents

There are f *weakly Byzantine* agents among k agents. Weakly Byzantine agents act arbitrarily without following an algorithm, but except changing their IDs. All agents except weakly Byzantine agents are called *good*. Good agents know neither the actual value nor the upper bound of f. The adversary wakes up at least one good agent at the first round.

(4) The gathering problems

We consider the following two problems. The gathering problem *with non-simultaneous termination* requires agents to satisfy the following conditions: (1) every good agent terminates an algorithm, and (2) when all the good agents terminate an algorithm, they are at the same node. The gather-

ing problem *with simultaneous termination* requires all the good agents to terminate an algorithm at the same round at the same node.

We measure the time complexity of a gathering algorithm by the number of rounds from beginning (i.e., the first good agent wakes up) to the round in which all the good agents terminate.

(5) Procedures

In the proposed algorithms, we use the graph exploration procedure and the extended label proposed in the literature.

The exploration procedure, called EXPLO(N), allows an agent to visit all nodes of any graph composed of at most N nodes, starting from any node of the graph. An implementation of this procedure is based on universal exploration sequences (UXS) and is a corollary of the result by Reingold [23]. The number of moves of EXPLO(N) is fixed not depending at the node where an agent starts this procedure. Hence, the number of moves of EXPLO(N), denoted by X_N, can be calculated locally with N.

Let $b_1 b_2 \cdots b_\ell$ be the binary representation of $a_i.ID$, where $\ell = \lfloor \log(a_i.ID) \rfloor + 1$. The extended label of a_i is defined as $a_i.ID^* = 10 b_1 b_1 b_2 b_2 \cdots b_\ell b_\ell \; 10 b_1 b_1 b_2 b_2 \cdots b_\ell b_\ell \cdots$. We have the following lemma about the extended label $a_i.ID^*$, which is used to prove the correctness of the proposed algorithms.

Lemma 2.1. *[6] For two different agents a_i and a_j, assume that $a_i.ID^* = x_1 x_2 \cdots$ and $a_j.ID^* = y_1 y_2 \cdots$ hold. Then, for some $k \leq 2\lfloor \log(\min(a_i.ID, a_j.ID)) \rfloor + 6$, $x_k \neq y_k$ holds.*

3. A Gathering Algorithm with Non-Simultaneous Termination

In this section, we propose an algorithm for the gathering problem *with non-simultaneous termination* by assuming a strong team composed of $4f^2 + 9f + 4$ agents. That is, at least $(4f+4)(f+1)$ good agents exist in the network. Recall that agents know N, but do not know n, k, or f.

3.1 Overview

The proposed algorithm aims to gather all good agents at a single node. The algorithm achieves this goal by three stages: CollectID, MakeGroup, and Gather stages. In the CollectID stage, agents collect IDs of all good agents and estimate the number of Byzantine agents at the end of the stage. In the MakeGroup stage, agents make a *reliable group*, which is composed of at least $4f + 4$ agents. In the Gather stage, all good agents gather at a single node and achieve the gathering. Each stage consists of multiple phases, and each phase consists of $P_N \geq X_N$ rounds. We will discuss the actual value of P_N later, and here just note that the duration of each phase is sufficient for an agent to explore the network by EXPLO(N). For simplicity, we first explain the overview under the assumption 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 $a_i.ID^*$ and decides the behavior. If the bit is 1, a_i executes EXPLO(N) during the phase. If the bit is 0, a_i waits during the phase. Agent a_i has variable $a_i.L$ to store a set of IDs, and if a_i finds another agent at the same node while exploring or waiting, it records the agent's 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 2.1, a_i can meet all other good agents and hence obtain IDs of all good agents.

In the MakeGroup stage, agents make a reliable group composed of at least $4f + 4$ agents. To do this, agents with small IDs keep waiting, and the other agents search for the agents with small IDs. More concretely, if the $f + 1$ smallest IDs in $a_i.L$ contains $a_i.ID$, a_i keeps waiting during this stage. Otherwise, a_i assigns 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 EXPLO(N) in a phase. If a_i finds a_{target} at some node, it ends the search and waits at the node. If a_i does not find a_{target} even after completing EXPLO(N), it regards a_{target} as a Byzantine agent. In this case, a_i assigns 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. Agent a_i continues this behavior until it finds a target agent. Since there are at most f Byzantine agents, the good agent with the smallest ID, say a_{min}, keeps waiting during the MakeGroup stage. This means that agents always find a_{min} if they search for a_{min}, and consequently, the number of agents searched for by good agents is at most $f + 1$ (including a_{min} and f Byzantine agents). Since at least $(4f+4)(f+1)$ good agents exist, even if the good agents are distributed to $f + 1$ nodes evenly, at least $4f+4$ agents gather in one node according to the pigeonhole principle. In other words, agents can make a reliable group. The ID of the target agent in a reliable group is used as the group ID. For Gather stage, a reliable group is divided into two groups, an exploring group and a waiting group, so that each of which contains 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, and agents collect group IDs of all reliable groups in the first phase of the Gather stage. More concretely, while agents in a waiting group keep waiting, other agents (in an exploring group or not in a reliable group) explore the network by EXPLO(N). When a_i finds a reliable group, it records the group ID. Note that, since each of an exploring group and a waiting group contains at least $2f + 2$ agents when it is created, it contains at least $f + 2$ good agents. Therefore, when an agent meets an exploring or waiting group in which at least $f + 2$ agents present the same group ID, the agent can understand that this group contains at least two good agents, and hence it is trustworthy. Note that, as we explain later, the proposed algorithm makes the group include at least two good agents because agent must actually use estimated values of f and the estimated values of f differ by at most one among good agents.

In the second phase of the GATHER stage, agents move to the node where the waiting group of the smallest group ID stays. That is, while agents in the waiting group of the smallest group ID keep waiting, other agents search for the group by EXPLO(N).

However, there are three problems to implement the above behavior.

The first problem is that agents not in a reliable group cannot instantly know the fact that a reliable group has been created, and so they do not know when to transition to the GATHER stage. To solve this problem, we make agents execute the MAKEGROUP stage and the GATHER stage alternately. Here, we design the two stages so that (1) agents achieve the gathering in the GATHER stage if a reliable group is created in the MAKEGROUP stage, and (2) otherwise 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, at the end of the COLLECTID stage, agents estimate the number of Byzantine agents, say $\tilde{f}$, from the fact that their ID lists 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 calculate the value of $\tilde{f}$ that satisfies $k \geq (4\tilde{f}+4)(\tilde{f}+1)$ and $f \leq \tilde{f}$. With these constraints, we ensure that good agents make at least one reliable group. However, values of $\tilde{f}$ differ by at most one among good agents, because some good agents may meet some Byzantine agents but others may not in the COLLECTID stage. Therefore, we design a method to make a reliable group such that each of the waiting group and the exploring group includes 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 EXPLO(N) to wake up dormant agents. As a result, we guarantee that all good agents start the algorithm within X_N rounds, but there still exists a problem. Good agents execute different phases at the same round because these agents woke up at different rounds. So, we adjust the number of rounds of each phase to guarantee that all the good agents execute the same phase at the same time for sufficient rounds.

3.2 Details

Algorithm 1 is the pseudocode of the proposed algorithm. The proposed algorithm realizes the gathering using three stages: The COLLECTID stage makes 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 EXPLO(N) to wake up all dormant agents (line 6 of Algorithm 1). By this behavior, after the first good agent wakes up, all good agents wake up within X_N rounds. After that, a_i executes phases of the COLLECTID, MAKEGROUP, and GATHER stages. Here we define one phase as $P_N = 3X_N + 1$ rounds. Since all good agents wake up within X_N rounds, the (X_N+1)-th to $2X_N$-th rounds of the x-th phase of good agent a_i overlap with the first $3X_N$ rounds of the x-th phases of all other good agents. Hence, we have the following observation.

Algorithm 1 Procedure ByzantineGathering(N) for an agent a_i whose $a_i.ID = b_1 b_2 \cdots b_\ell$ 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
6:  Explore the network by EXPLO(N)
7:  while True do
8:      if a_i.EndCI = False then
9:          Execute a_i.x-th phase of the CollectID stage
10:     else
11:         Execute the MakeGroup stage
12:     end if
13:     a_i.x ← a_i.x + 1
14:     Execute the Gather stage
15: end while
```

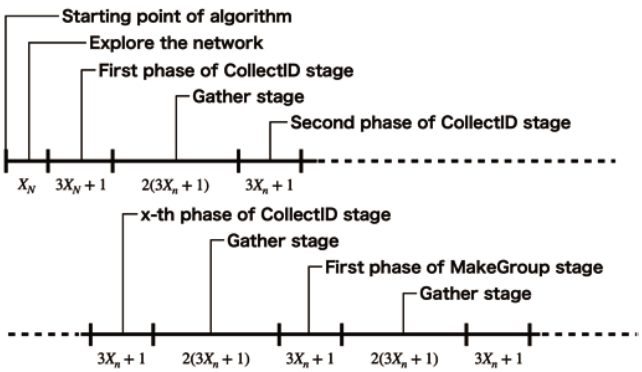


Fig. 1 The stage flow.

Observation 3.1. *Let a_i and a_j be good agents. Assume that a_i explores the network with EXPLO(N) from the (X_N+1)-th round to the $2X_N$-th round of its x-th phase, and a_j waits during the first $3X_N$ 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 9 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 alternately executes one phase of the MAKEGROUP stage (instead of the COLLECTID stage) and two phases of the GATHER stage (lines 11 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.

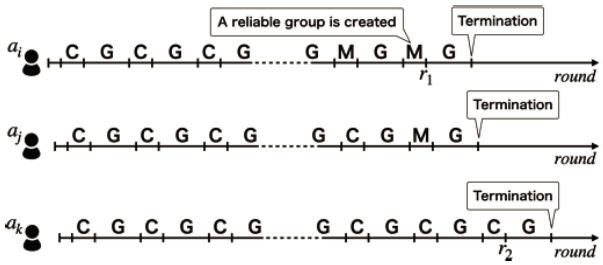


Fig. 2 An example execution of the algorithm by good agents a_i, a_j, and a_k.

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 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 2 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.

3.2.1 The 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 $3X_N$ rounds in the x-th phase (lines 1–2 of Algorithm 2). If the x-th bit is 1, a_i waits for X_N rounds, explores the network by EXPLO(N), and then waits for X_N 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 2.1 and Observation 3.1, 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 12). As we prove later, $a_i.\tilde{f} \geq f$ holds in Lemmas 3.2 and 3.3, 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 14).

Table 2 Variables of agents.

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.

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

```
1: if the a_i.x-th bit of a_i.ID* is 0 then
2:     Wait for 3X_N rounds at the current node
3:     a_i.L ← a_i.L ∪ {IDs of agents a_i met while waiting}
4: else
5:     Wait for X_N rounds at the current node
6:     Explore the network by EXPLO(N)
7:     Wait for X_N rounds at the current node
8:     a_i.L ← a_i.L ∪ {IDs of agents a_i met while exploring}
9: end if
10: // The (3X_N + 1)-th round
11: if a_i.x = 2⌊log(a_i.ID)⌋ + 6 then
12:     a_i.f̃ ← max{y | (4y + 4)(y + 1) ≤ |a_i.L|}
13:     a_i.x ← 1
14:     a_i.EndCI ← True
15: end if
16: Wait for one round
```

Algorithm 3 MakeGroup stage for an agent a_i

```
1: if a_i.x = 1 then
2:     if the smallest a_i.f̃ + 1 IDs in a_i.L contain a_i.ID then
3:         a_i.state ← TargetAgent
4:     else
5:         a_i.state ← SearchAgent
6:     end if
7: end if
8: if a_i.state = TargetAgent then
9:     //a_i is a target agent
10:    a_i.target ← a_i.ID
11:    Wait for one phase at the current node
12:    and
13:    While waiting, execute consensus() every round
14: else
15:    //a_i is a search agent
16:    a_i.target ← min(a_i.L \ a_i.BL)
17:    Wait for X_N rounds at the current node
18:    Search for an agent a_target with ID a_i.target by EXPLO(N)
19:    and
20:    if a_i meets a_target while searching then
21:        Stop EXPLO(N)
22:        Wait until the end of the phase
23:        and
24:        While waiting, execute consensus() every round
25:        and
26:        if a_i finds a_target Byzantine while waiting then
27:            // This is true if, during the (X_N + 1)-th round to
28:            // the 2X_N-th round, a_target moved to another
29:            // node or a_target.target ≠ a_target.ID holds
30:            a_i.BL ← a_i.BL ∪ {a_i.target}
31:        end if
32:    else
33:        // a_i does not meet a_target and hence a_target is Byzantine
34:        a_i.BL ← a_i.BL ∪ {a_i.target}
35:        Wait until the end of the phase
36:    end if
37: end if
```

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 · a_i.f̃ then
2:     a_i.F ← the most frequent value of f̃ 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| ≥ 4 · a_i.F + 4 and there exists a_target with a_target.target =
   a_target.ID = a_i.target then
5:         a_i.GID ← a_target.ID
6:         if the 2 · a_i.F + 2 smallest IDs in GC contain a_i.ID then
7:             a_i.state ← ExploringGroup
8:         else
9:             a_i.state ← WaitingGroup
10:        end if
11:    end if
12: end if
```

3.2.2 The 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 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 3.1 (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 10) and waits for one phase at the current node (line 11). While waiting, a_i executes procedure consensus() to create a reliable group if possible (line 13). 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 $a_i.BL$ (line 16). After that, a_i waits for X_N rounds and then searches for the agent with ID $a_i.target$, say a_{target}, by executing EXPLO(N) (lines 17 and 18). If a_i finds a_{target} at the same node during the exploration, a_i ends EXPLO(N) and waits at the node until the end of the phase (lines 21–22). 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 33–34), or (2) after a_i found a_{target}, during the $(X_N + 1)$-th round to the $2X_N$-th round, a_{target} moved to another node or $a_{target}.target \neq a_{target}.ID$ holds (lines 26–30). In this case, a_i adds $a_{target}.ID$ to $a_i.BL$, and never searches for a_{target} in the later phases of the MakeGroup stage (lines 30 and 34). If a_i did not find a_{target}, it waits until the end of the phase (line 35).

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 mul-

tiple 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 3.2 and 3.6) (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 3.3). 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.

3.2.3 The 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 1–2 of Algorithm 5). In the following, we describe the behaviors of agents that have finished the CollectID stage.

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 ≠ NULL, execute a_i.GL ← a_i.GL ∪ {(a_j.GID, a_j.ID)}
9:     else
10:        Wait for X_N rounds at the current node
11:        Explore the network by EXPLO(N)
12:        and
13:        While exploring, whenever a_i meets a_j with a_j.GID ≠ NULL, execute a_i.GL ← a_i.GL ∪ {(a_j.GID, a_j.ID)}
14:        Wait for X_N + 1 rounds at the current node
15:    end if
16:    // The second phase
17:    //MemberID(gid) = {id | (gid, id) ∈ a_i.GL}
18:    //ReliableGID() = {gid | |MemberID(gid)| ≥ a_i.f̃ + 1}
19:    if ReliableGID() = ∅ then
20:        Wait for one phase at the current node
21:    else if a_i.state = WaitingGroup and a_i.GID = min(ReliableGID()) then
22:        Wait for 3X_N rounds at the current node
23:        Terminate the algorithm
24:    else
25:        Wait for X_N rounds at the current node
26:        By executing EXPLO(N), search for the node with a reliable waiting group whose group ID is min(ReliableGID())
27:        Wait at the node until the last round of the phase
28:        Terminate the algorithm at the last round of the phase
29:    end if
30: end if
```

If agents have finished the CollectID stage, they try to achieve the gathering in two phases of the Gather stage. In the first phase of the two phases, agents collect group IDs of all reliable groups (lines 4–15). 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 $(X_N + 1)$-th round to the $2X_N$-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 16–29).

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 3.3 and 3.6). 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 X_N rounds, explores the network, and then waits for $X_N + 1$ 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., $MemberID(gid) = \{id \mid (gid, id) \in a_i.GL\}$), and (2) a_i checks up group IDs such that at least $a_i.\tilde{f} + 1$ agents conveyed the group ID (i.e., $ReliableGID() = \{gid \mid |MemberID(gid)| \geq a_i.\tilde{f} + 1\}$). Note that, if a_i belongs to a exploring (resp., waiting) group, $a_i.GID \in ReliableGID()$ 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 $a_i.GID = \min(ReliableGID())$, it waits for $3X_N$ rounds and terminates the algorithm (lines 21–23). Otherwise, a_i waits for X_N rounds and searches for the node with the waiting group whose group ID is $\min(ReliableGID())$ by executing `EXPLO`(N) (lines 25–26). After that, a_i waits until the last round of this phase and terminates the algorithm at the node (lines 27–28).

3.3 Correctness and Complexity

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

Lemma 3.1. *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 2.1 and Observation 3.1, 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 3.2. *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 3.1, 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 3.3. *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 $(4(p+2)+4)((p+2)+1) \leq |a_j.L|$ by $a_j.\tilde{f} \geq p+2$. Therefore, since $p \geq f$ holds by Lemma 3.2, $|a_j.L| - |a_i.L| > 8p+20 > f$ holds. On the other hand, since $a_i.L$ and $a_j.L$ include IDs of all good agents by Lemma 3.1, we have $|a_j.L| - |a_i.L| \leq f$, 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 3.4. *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 3.2, $a_{min}.\tilde{f} \geq f$ holds; thus, the $a_{min}.\tilde{f} + 1$ ($\geq f + 1$) 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 3.5. *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 34 of Alg. 3).
2. After a_i found a_{target}, during the $(X_N + 1)$-th round to the $2X_N$-th round of the phase, a_{target} moved to another node or $a_{target}.target \neq a_{target}.ID$ holds (line 30 of Alg. 3).

For the base case, we consider the first phase of the MAKEGROUP stage of a_i. By Lemma 3.1, $a_i.L$ 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 $(X_N + 1)$-th round of the first phase of the MakeGroup stage. Since a_{min} is a target agent by Lemma 3.4, 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 3.6. *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 3.2, 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 3.3, 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 3.7. *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 3.3 and 3.6, $a_j.\tilde{f} \leq a_i.F + 1$ 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 3.2, 3.3 and 3.6, 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 3.1.

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 X_N rounds and each phase consists of $3X_N + 1$ rounds, any good agent a_i has exactly one phase $phase_x^i$ that overlaps $phase_x$ for at least $2X_N + 1$ 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 3.8. *Let $Byz_1, Byz_2, \ldots, Byz_{f'}$ ($Byz_l.ID < Byz_{l+1}.ID$ for $1 \leq l \leq f' - 1$) 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 ($1 \leq x \leq f'$) 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 $2X_N$-th round of the first phase. In addition, they start the first phase within at most X_N round and wait during the $(2X_N + 1)$-th round to the $(3X_N + 1)$-th round of the first phase. This implies that all

agents in A_1 exist at the same node as Byz_1 before the $2X_N$-th round of the first phase, and at that time the number of agents at the node is at least $|A_1 \cup \{Byz_1\}| \geq 4\tilde{f}_{max} + 4$. 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 3.7, since a reliable group is created by the algorithm, this contradicts the assumption. Therefore, $|A_1| \leq 4\tilde{f}_{max} + 2$ holds.

For induction step, assume that, in the $(x+1)$-th phase $(1 \leq x < f')$, 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 $bid \in \{Byz_1.ID, Byz_2.ID, \ldots, Byz_x.ID\}$ to *target* in the $(x+1)$-th phase. Since good agents monotonically increase *target*, $A_x \subseteq B_x \cup C_x$ holds. Since $|C_x| \leq (4\tilde{f}_{max} + 2)x$ holds by the assumption of induction, $|B_x \cap A_x| \geq |A_x| - |C_x| > 4\tilde{f}_{max} + 2$ 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 $2X_N$-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 3.7, since a reliable group is created by the algorithm, this contradicts the assumption. Therefore, $|A_x| \leq (4\tilde{f}_{max} + 2)(x+1)$ holds.

Hence, the lemma holds. □

Lemma 3.9. *Before a_{last} finishes the $(f+1)$-th phase of the* MakeGroup *stage, a reliable group is created.*

Proof. Let $f' (\leq f)$ be the number of Byzantine agents whose IDs are smaller than $a_{min}.ID$. By Lemma 3.8, 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 3.4, 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 3.5). Since they can successfully find a_{min}, by Lemma 3.2, at least $(k-f) - (\tilde{f}_{max} + 1) - (4\tilde{f}_{max} + 2)f' \geq (4\tilde{f}_{max} + 4)(\tilde{f}_{max} + 1) - \tilde{f}_{max} - (\tilde{f}_{max} + 1) - (4\tilde{f}_{max} + 2)\tilde{f}_{max} = 4\tilde{f}_{max} + 3$ search agents stay with target agent a_{min} before the $2X_N$-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 3.7, 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 X_N 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 $r + X_N$. We define $Rel(r)$ as a set of reliable groups that exist in round $r + X_N$. 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 3.10. *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 $r \leq r' \leq r + X_N$ 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 X_N rounds, all good agents start two consecutive phases of the Gather stage during rounds r to $r + X_N$ and hence, no new reliable group is created during rounds $r + X_N$ to $r + 2X_N$.

If a_i belongs to a waiting group, it waits during rounds $r' (\leq r + X_N)$ to $r' + 3X_N (\geq r + 3X_N)$. Since all good agents in exploring groups of $Rel(r)$ explore the network during rounds $r + X_N$ to $r + 3X_N$, 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 $r' + X_N (\geq r + X_N)$ to $r' + 2X_N (\leq r + 3X_N)$. Since all good agents in waiting groups of $Rel(r)$ wait during rounds $r + X_N$ to $r + 3X_N$, 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 3.7, each of the exploring group and the waiting group of G contains at least $\tilde{f}_{min} + 2$ good agents. By Lemma 3.3, 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 3.11. *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 $r + X_N$. Assume that a_i (possibly a_{ini}) starts two consecutive phases of the* Gather *stage in round r' such that $r \leq r' \leq r + X_N$. 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 3.10, 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 $(3X_N + 1)$-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 $(X_N + 1)$-th round to the $2X_N$-th round in the second phase. Recall that agents in a waiting group of $G_{min}(r)$ wait for $3X_N$ rounds before terminating at $v_{min}(r)$ in their second phases, and the difference of starting times of the phases is at most X_N. 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, $gid_{min}(r) = \min(ReliableGID())$ 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 3.1. *Let n be the number of nodes, k be the number of agents, f be the number of weakly Byzantine agents, and Λ_{good} be the largest ID among good 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 $X_N + 3(2\lfloor\log(\Lambda_{good})\rfloor + f + 7)(3X_N + 1)$ rounds.*

Proof. Let a_{last} be the good agent that finishes the CollectID stage last. Since a_{last} wakes up within X_N 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 $X_N + (2\lfloor\log(\Lambda_{good})\rfloor + 6) \cdot 3(3X_N + 1) = X_N + 3(2\lfloor\log(\Lambda_{good})\rfloor + 6)(3X_N + 1)$ rounds. By Lemma 3.9, a reliable group is created before a_{last} finishes the $(f + 1)$-th phase of the MakeGroup stage. By Lemma 3.11, 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 $X_N + 3(2\lfloor\log(\Lambda_{good})\rfloor + 6)(3X_N + 1) + (f + 1) \cdot 3(3X_N + 1) = X_N + 3(2\lfloor\log(\Lambda_{good})\rfloor + f + 7)(3X_N + 1)$ rounds. □

4. A 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 3.11, 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 $r_i + X_N$ because all agents wake up within X_N 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 $T = X_N + X_N + 3(2\lfloor\log(a_i.ID_{max})\rfloor + 6)(3X_N + 1)$ rounds after a_i starts the algorithm. For this reason, a_i sets $a_i.flag_t \leftarrow True$ if (a) X_N rounds have elapsed after round r_i and (b) T rounds have elapsed after it started the algorithm.

Theorem 4.1. *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 $4f^2+9f+4 \leq k$ holds, the proposed algorithm solves the gathering problem with simultaneous termination in at most $3X_N + 3(2\lfloor\log(\Lambda_{all})\rfloor + f + 7)(3X_N + 1) + 1$ rounds.*

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 $r + X_N$, and let $Rel(r)$ be a set of reliable groups that exist in round $r + X_N$. 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 3.11, 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 $r_f + X_N$. Since $r^* \geq r_f + X_N$ 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 X_N rounds and agents do not move during the last X_N 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 3.6, 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 $flag_t \leftarrow True$.

Lastly, we prove that good agents terminate in at most $3X_N + 3(2\lfloor\log(\Lambda_{all})\rfloor + f + 7)(3X_N + 1) + 1$ rounds. Similarly to Theorem 3.1, all good agents complete the previous algorithm and gather at $v_{min}(r)$ in at most $T_1 = X_N + 3(2\lfloor\log(\Lambda_{good})\rfloor + f + 7)(3X_N + 1)$ rounds. In addition, since ID_{max} is an ID of some agent, good agents wait until at most $T_2 = 2X_N + 3(2\lfloor\log(\Lambda_{all})\rfloor + 6)(3X_N + 1)$ rounds have passed. Note that good agents execute $flag_t \leftarrow True$ if (a) X_N 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 $T_3 = max\{T_1 + X_N, T_2\} \leq 2X_N + 3(2\lfloor\log(\Lambda_{all})\rfloor + f + 7)(3X_N + 1)$ rounds after they start the algorithm. Since all good agents start the algorithm within X_N rounds and they terminate after at least $F + 1$ agents execute $flag_t \leftarrow True$, they terminate in at most $X_N + T_3 + 1 = 3X_N + 3(2\lfloor\log(\Lambda_{all})\rfloor + f + 7)(3X_N + 1) + 1$ rounds after the first good agent wakes up. □

5. Conclusion

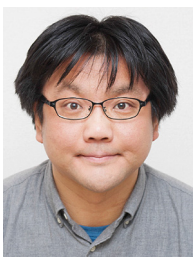
In this paper, we have developed two algorithms that achieve the gathering in weakly Byzantine environments. We proposed two algorithms that reduce the time complexity compared to the existing algorithm by assuming a strong team

of agents. The proposed algorithms operate under the assumption that the upper bound N of the number of nodes is given to agents, and at least $(4f+4)(f+1)$ good agents exist in the network, where f is the number of Byzantine agents. The first algorithm achieves the gathering with non-simultaneous termination in $O((f+|\Lambda_{good}|)\cdot X(N))$ rounds, where $|\Lambda_{good}|$ is the length of the largest ID among good agents and $X(N)$ is the number of rounds required to explore any network composed of at most N nodes. The second algorithm achieves the gathering with simultaneous termination in $O((f+|\Lambda_{all}|)\cdot X(N))$ rounds, where $|\Lambda_{all}|$ is the length of the largest ID among agents. As a future work, it would be interesting to study the trade-off between the time complexity and the ratio of good and Byzantine agents.

References

[1] A. Pelc, "Deterministic rendezvous algorithms," Distributed Computing by Mobile Entities, Current Research in Moving and Computing, ed. P. Flocchini, G. Prencipe, and N. Santoro, pp.423–454, Springer, 2019.
[2] Y. Dieudonné, A. Pelc, and D. Peleg, "Gathering Despite Mischief," ACM Transactions on Algorithms, vol.11, no.1, pp.1–28, 2014.
[3] S. Bouchard, Y. Dieudonné, and B. Ducourthial, "Byzantine gathering in networks," Distributed Computing, vol.29, no.6, pp.435–457, 2016.
[4] S. Bouchard, Y. Dieudonné, and A. Lamani, "Byzantine gathering in polynomial time," 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, pp.147:1–147:15, 2018.
[5] T. Schelling, The Strategy of Conflict, Harvard University Press, 1960.
[6] A. Dessmark, P. Fraigniaud, D.R. Kowalski, and A. Pelc, "Deterministic rendezvous in graphs," Algorithmica, vol.46, no.1, pp.69–96, 2006.
[7] D.R. Kowalski and A. Malinowski, "How to meet in anonymous network," Theoretical Computer Science, vol.399, no.1-2, pp.141–156, 2008.
[8] A. Miller and A. Pelc, "Time versus cost tradeoffs for deterministic rendezvous in networks," Distributed Computing, vol.29, no.1, pp.51–64, 2016.
[9] A. Ta-Shma and U. Zwick, "Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences," ACM Transactions on Algorithms, vol.10, no.3, pp.12:1–12:15, 2014.
[10] J. Czyzowicz, A. Kosowski, and A. Pelc, "How to meet when you forget: log-space rendezvous in arbitrary graphs," Distributed Computing, vol.25, no.2, pp.165–178, 2012.
[11] P. Fraigniaud and A. Pelc, "Delays induce an exponential memory gap for rendezvous in trees," ACM Transactions on Algorithms, vol.9, no.2, pp.17:1–17:24, 2013.
[12] P. Fraigniaud and A. Pelc, "Deterministic rendezvous in trees with little memory," Distributed Computing, 22nd International Symposium, DISC 2008, pp.242–256, 2008.
[13] Y. Dieudonné, A. Pelc, and V. Villain, "How to meet asynchronously at polynomial cost," SIAM Journal on Computing, vol.44, no.3, pp.844–867, 2015.
[14] S. Guilbault and A. Pelc, "Gathering asynchronous oblivious agents with local vision in regular bipartite graphs," Theoretical Computer Science, vol.509, pp.86–96, 2013.
[15] E. Kranakis, D. Krizanc, E. Markou, A. Pagourtzis, and F. Ramírez, "Different speeds suffice for rendezvous of two agents on arbitrary graphs," Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2017, pp.79–90, 2017.
[16] G.D. Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro, "Asynchronous deterministic rendezvous in graphs," Theoretical Computer Science, vol.355, no.3, pp.315–326, 2006.
[17] E. Bampas, J. Czyzowicz, L. Gasieniec, D. Ilcinkas, and A. Labourel, "Almost optimal asynchronous rendezvous in infinite multidimensional grids," Distributed Computing, 24th International Symposium, DISC 2010, pp.297–311, 2010.
[18] A. Collins, J. Czyzowicz, L. Gasieniec, and A. Labourel, "Tell me where I am so I can meet you sooner," Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, pp.502–514, 2010.
[19] J. Czyzowicz, A. Pelc, and A. Labourel, "How to meet asynchronously (almost) everywhere," ACM Transactions on Algorithms, vol.8, no.4, pp.37:1–37:14, 2012.
[20] M. Tsuchida, F. Ooshita, and M. Inoue, "Byzantine-tolerant gathering of mobile agents in arbitrary networks with authenticated whiteboards," IEICE Transactions, vol.E101-D, no.3, pp.602–610, 2018.
[21] M. Tsuchida, F. Ooshita, and M. Inoue, "Byzantine-tolerant gathering of mobile agents in asynchronous arbitrary networks with authenticated whiteboards," IEICE Transactions on Information and Systems, vol.E103-D, no.7, pp.1672–1682, 2020.
[22] A. Miller and U. Saha, "Fast byzantine gathering with visibility in graphs," Algorithms for Sensor Systems - 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2020, Springer, 2020.
[23] O. Reingold, "Undirected connectivity in log-space," Journal of the ACM, vol.55, no.4, pp.17:1–17:24, 2008.

Jion Hirose received the B.E. degree from Toyohashi University of Technology, Japan in 2019 and the M.E. degree from Nara Institute Science and Technology, Japan in 2021. He is now a student at the Graduate School of Science and Technology, Nara Institute Science and Technology. His research interests include distributed algorithms.



Junya Nakamura received the B.E. and M.E. degrees from Toyohashi University of Technology, Japan in 2006 and 2008, respectively, and the Ph.D. degree in Information Science and Technology from Osaka University in 2014. He is currently an associate professor of Information and Media Center, Toyohashi University of Technology. His research interests include theoretical and practical aspects of distributed algorithms and systems. He is a member of IEEE, IEICE, and IPSJ.

Fukuhito Ooshita received his M.E. and D.I. degrees in computer science from Osaka University in 2002 and 2006, respectively. He was an assistant professor at the Graduate School of Information Science and Technology at Osaka University during from 2003 to 2015. He is now an associate professor at the Graduate School of Science and Technology, NAIST. His research interests include parallel and distributed algorithms. He is a member of ACM, IEEE, IEICE, and IPSJ.

Michiko Inoue received her B.E., M.E, and Ph.D. degrees in Computer Science from Osaka University in 1987, 1989, and 1995 respectively. She worked at Fujitsu Laboratories Ltd. from 1989 to 1991. She is a Professor of Graduate School of Science and Technology, Nara institute of Science and Technology (NAIST). Her research interests include distributed algorithms, graph theory and dependability of integrated circuits. She is a fellow of IEICE, a senior member of IEEE, and a member of IPSJ and JSAI.