Gathering with a strong team in weakly Byzantine environments

Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue

1Nara Institute of Science and Technology
2Toyohashi University of Technology

July 15, 2020

Abstract

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 \(O(n^4 \cdot |\Lambda_{\text{good}}| \cdot X(n))\) rounds, where \(|\Lambda_{\text{good}}|\) 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 and all the agents awake at the same time. Both the algorithms take the upper bound \(N\) of \(n\) as input. The first algorithm achieves gathering with non-simultaneous termination in \(O((f + |\Lambda_{\text{good}}|) \cdot X(N))\) rounds. The second algorithm achieves gathering with simultaneous termination in \(O((f + |\Lambda_{\text{all}}|) \cdot X(N))\) rounds, where \(|\Lambda_{\text{all}}|\) 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 the existing one if \(|\Lambda_{\text{all}}| = O(|\Lambda_{\text{good}}|)\) holds.

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 task that collects multiple agents on the same node is called a gathering, and this task has been widely studied from the theoretical aspect of distributed systems [1]. By accomplishing this task, the agents can exchange information with each other more efficiently, and it becomes easy to 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 study, we consider the deterministic gathering problem with Byzantine agents and propose two 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 agent synchronization, anonymity, presence/absence of memory on a node (called whiteboard), presence/absence of randomization, and topology, etc. \cite{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, and memory space, etc.) if solvable. The rest of this section describes 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 \cite{1}. These works have studied the feasibility of the rendezvous and, if feasible, the time required to accomplish the task. 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 \cite{3,4,5,6}, rendezvous algorithms have been proposed in any graph by assuming a unique ID for each agent. Dessmark et al. \cite{6} have proposed an algorithm to achieve the rendezvous in polynomial time of \(n\), \(\lambda\) and \(\tau\), where \(n\) is the number of nodes, \(\lambda\) is the smallest ID among agents, and \(\tau\) is the difference between the startup times of agents. Kowalski et al. \cite{7} and Ta-shma et al. \cite{8} have improved the time complexity and have proposed algorithms to achieve the rendezvous in time independent of \(\tau\). In addition, Millar et al. \cite{9} have analyzed the trade-off between the time required for rendezvous and the number of moves. On the other hand, some papers \cite{10,11,12} 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, they have proposed algorithms for limited graphs and initial arrangements. Fraigniaud et al. \cite{10,11} have proposed algorithms for trees, and Czyzowicz et al. \cite{12} 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 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 \cite{13,14,15,16} for a finite graph and the literature \cite{17,18,19} for an infinite graph.

Recently some papers \cite{2,3,20,21} have studied the gathering problem in the presence of Byzantine agents. Table \cite{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. \cite{2} have studied the gathering problem in synchronous environments where \(k\) agents exist in a \(n\)-node arbitrary network and \(f\) of them are Byzantine. For weakly Byzantine agents, if \(n\) is given to agents, the gathering algorithm with the time complexity of \(O(n^4 \cdot |\Lambda_{good}| \cdot X(n))\) has been proposed, where \(|\Lambda_{good}|\) 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, while, if \(f\) is given to agents, the gathering algorithm with the time complexity that is polynomial of \(n\) and \(|\Lambda_{good}|\)

Table 1: A summary of synchronous Byzantine gathering algorithms with unique IDs. Here, \(n\) is the number of nodes, \(N\) is the upper bound of \(n\), \(|\Lambda_{good}|\) is the length of the smallest ID among good agents, \(|\Lambda_{bad}|\) is the length of the largest ID among good agents, \(|\Lambda_{dis}|\) is the length of the largest ID among agents, \(k\) is the number of agents, and \(f\) is the number of Byzantine agents.

| Input | Startup delay | Byzantine | Condition of weak Byzantine agent | # of Byzantine agents | Time complexity |
|-------|--------------|-----------|-------------------------------|-----------------------|----------------|
| \(n\) | \(n\) | Possible | Weak | \(f + 1 \leq k\) | Possible | \(O(n^4 \cdot |\Lambda_{good}| \cdot X(n))\) |
| \(f\) | | Possible | Weak | \(2f + 2 \leq k\) | Possible | \(O(\text{Poly.} \cdot n \cdot |\Lambda_{good}|)\) |
| \(n,f\) | \(|\log\log n|\) | Possible | Strong | \(2f + 1 \leq k\) | Possible | \(O(\text{Exp.} \cdot n \cdot |\Lambda_{good}|)\) |
| \(f\) | | Possible | Strong | \(5f^2 + 7f + 2 \leq k\) | Possible | \(O(\text{Poly.} \cdot n \cdot |\Lambda_{good}|)\) |

Proposed algorithm 1 | \(N\) | Impossible | Weak | \(4f^2 + 9f + 4 \leq k\) | Impossible | \(O((f + |\Lambda_{good}|) \cdot X(N))\) |

Proposed algorithm 2 | \(N\) | Impossible | Weak | \(4f^2 + 9f + 4 \leq k\) | Possible | \(O((f + |\Lambda_{good}|) \cdot X(N))\) |
has been proposed. The numbers of non-Byzantine agents required for the gathering algorithms are at least one and \( f + 2 \), respectively, and numbers are proven to be tight. On the other hand, for strongly Byzantine agents, in the cases where \( n \) and \( f \) are given to agents and \( f \) is given to agents, the gathering algorithms whose time complexities are exponential of \( n \) and \( \left| \Lambda_{\text{good}} \right| \) have been proposed. The numbers of non-Byzantine agents required for the gathering algorithms are at least \( 2f + 1 \) and \( 4f + 2 \), respectively, while the numbers of non-Byzantine agents required to solve the gathering problems under these conditions are \( f + 1 \) and \( f + 2 \), respectively. Bouchard et al. \[3\] have proposed the algorithms that show tight results for the number of non-Byzantine agents required to solve the gathering problem for both cases in the presence of strongly Byzantine agents. That is, the numbers of non-Byzantine agents required for the algorithms are at least \( f + 1 \) and \( f + 2 \), respectively. However, the time complexities of the algorithms are still exponential of \( n \).

Bouchard et al. \[4\] have proposed the gathering algorithm with the time complexity that is polynomial time for the first time in presence of strongly Byzantine agents in synchronous environments. The gathering algorithm operates under the assumption that \( \log \log n \) is given to agents and at least \( 5f^2 + 6f + 2 \) non-Byzantine agents exist in the network.

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 \( f \) 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, the existing fastest algorithm is the one proposed by Dieudonné et al. \[2\]. The algorithm tolerates any number of weakly Byzantine agents, achieves the gathering with simultaneous termination, and its time complexity is \( O(n^4 \cdot \left| \Lambda_{\text{good}} \right| \cdot X(n)) \), where \( n \) is the number of nodes, \( \left| \Lambda_{\text{good}} \right| \) 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. When agents can use authenticated whiteboards on nodes, Tsuchida et al. \[20\] have proposed the algorithm that is faster than that of Dieudonné et al. \[2\]. However, the assumptions of authenticated whiteboards are strong and greatly restrict the behavior of Byzantine agents.

In this paper, we try to reduce the time complexity by taking advantage of a strong team, that is, a team 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 and all the agents awake at the same time (see Table \[1\]. In the algorithms, agents know the upper bound \( N \) of \( n \) and start the algorithms at the same time. The first algorithm achieves the gathering with non-simultaneous termination and its time complexity is \( O((f + \left| \Lambda_{\text{good}} \right|) \cdot X(N)) \), where \( \left| \Lambda_{\text{good}} \right| \) 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 \( O((f + \left| \Lambda_{\text{all}} \right|) \cdot X(N)) \), where \( \left| \Lambda_{\text{all}} \right| \) is the length of the maximum ID of all agents.

### 2 Preliminaries

**A 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 connected
to node \( v \) is locally and uniquely labeled by function \( P_v : \{ \{ v, u \} | u \in N_v \} \rightarrow \{ 1, 2, \ldots, d(v) \} \) 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 \( v \). Any node has neither ID nor memory. Time is discretized, and each discretized time is called a round.

**Mobile agents.** There are \( k \) agents \( a_1, a_2, \ldots, a_k \) in the system. At the beginning of an execution, the agents are scattered on the nodes and awake at the same time. All 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.

An agent is modeled as a state machine \((S, \delta)\). 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 \( \delta \) 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 on the same node, the degree of the current node, and the entry port. Agents execute the state transition synchronously at every round, and if an agent leaves the current node, the agent arrives at a destination node just before the next round. When an agent enters a node \( v \) via an edge \( \{ u, v \} \), it learns the degree \( d(v) \) of \( v \) and the port number \( P_v(v, u) \). 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 on node \( v \) at the same round, 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.

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

We measure the time complexity of a gathering algorithm by the number of rounds from beginning to the round in which all the good agents terminate.

**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 traverse 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 [22]. The number of moves of \( EXPLO(N) \) is denoted by \( X_N \).

Let \( b_1 b_2 \cdots b_{|a_i.ID|} \) be the binary representation of \( a_i.ID \), where \( |a_i.ID| = |a_i.ID| \). The extended label of \( a_i \) is defined as \( a_i.ID^* = 10b_1b_2b_2 \cdots b_{|a_i.ID|}0b_1b_2b_2 \cdots b_{|a_i.ID|} \). 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.** [2] For two different agents \( a_i \) and \( a_j \), assume that \( a_i.ID^* = x_1x_2 \cdots \) and \( a_j.ID^* = y_1y_2 \cdots \) hold. Then, for some \( k \leq 2[\log(\min(a_i.ID, a_j.ID))] + 6 \), \( x_k \neq y_k \) holds.
3 An algorithm for the gathering problem 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 on 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. 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 on a single node and achieve the gathering. For simplicity, we first explain the overview under the assumption that agents know \(f\).

In the CollectID stage, agents collect IDs of all good agents. To do this, each agent \(a_i\) reads bits of \(a_i.ID^*\) from the beginning. If the bit is 1, \(a_i\) executes \(\text{EXPLO}(N)\). If the bit is 0, \(a_i\) waits for \(X_N\) rounds (that is, rounds required for \(\text{EXPLO}(N)\)). Agent \(a_i\) has variable \(a_i.L\) to store a set of IDs, and if \(a_i\) finds another agent on the same node while exploring or waiting, it records the agent’s ID in \(a_i.L\). Agent \(a_i\) executes this procedure until it reads the \((2\lceil \log(a_i.ID) \rceil + 6)\)-th bit, 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 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 \(a_{\text{target}}\), by executing \(\text{EXPLO}(N)\). If \(a_i\) finds \(a_{\text{target}}\) on some node, it ends the search and waits on the node. If \(a_i\) does not find \(a_{\text{target}}\) even after completing \(\text{EXPLO}(N)\), it regards \(a_{\text{target}}\) 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 \(a_i.target\). Agent \(a_i\) continues this behavior until it finds the target agent. Since there are at most \(f\) Byzantine agents, the good agent with the smallest ID, say \(a_{\text{min}}\), keeps waiting during the MakeGroup stage. This means that agents always find \(a_{\text{min}}\) if they search for \(a_{\text{min}}\), and consequently, the number of agents searched for by good agents is at most \(f + 1\) (including \(a_{\text{min}}\) 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, and 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. To do this, agents first collect group IDs of all reliable groups. 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 \(\text{EXPLO}(N)\). 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, it contains at least \(f + 2\) good agents. Therefore, when another agent meets an exploring or waiting group, the agent can understand that this group contains at least two good agents, and hence it is reliable. After collecting group IDs, 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 \(\text{EXPLO}(N)\).

However, there are two 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.
Algorithm 1 Proposed algorithm$(N)$ for an agent $a_i$ whose $a_i.ID = b_1b_2 \cdots b_\ell$ where $\ell = |a_i.ID|$.

1: $a_i.state ← \text{CorrectID}$
2: $a_i.L ← \{a_i.ID\}$, $a_i.BL ← \emptyset$, $a_i.GL ← \emptyset$
3: $a_i.GID ← \text{NULL}$
4: $a_i.EndCI ← False$
5: $a_i.x ← 1$
6: while True do
7:   if $a_i.EndCI = False$ then
8:       Execute the $a_i.x$-th phase of the $\text{CollectID}$ stage
9:   else
10:      Execute the $\text{MakeGroup}$ stage
11: end if
12:   $a_i.x ← a_i.x + 1$
13: Execute the $\text{Gather}$ stage
14: end while

in the $\text{MakeGroup}$ stage, and (2) otherwise behaviors in the $\text{Gather}$ stage do not affect the $\text{MakeGroup}$ stage. The second problem is that agents do not know $f$. To solve this problem, at the end of the $\text{CollectID}$ stage, agents calculate the estimated number of Byzantine agents, say $\tilde{f}$, from the fact that at least $(4f + 4)(f + 1)$ good agents exist, and their ID lists include IDs of all good agents. 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 $\text{CollectID}$ stage. Therefore, we design the behaviors of the $\text{MakeGroup}$ stage and the $\text{Gather}$ stage so that agents can gather even if the estimated values have the difference.

3.2 Details

Algorithm 1 gives the pseudo-code of the main procedure of the proposed algorithm. The proposed algorithm realizes the gathering using three stages: The $\text{CollectID}$ stage makes agents collect IDs of all good agents, the $\text{MakeGroup}$ stage creates a reliable group composed of at least $4f + 4$ agents, and the $\text{Gather}$ stage gathers all good agents.

One phase is defined as continuous $X_N + 1$ rounds, and each stage is divided into multiple phases, as shown in Fig.1. After starting the algorithm, agent $a_i$ alternately executes one phase of the $\text{CollectID}$ stage and two phases of the $\text{Gather}$ stage (lines 8 and 13 of Algorithm 1). After $a_i$ finishes the $\text{CollectID}$ stage, it alternately executes one phase of the $\text{MakeGroup}$ stage (instead of the $\text{CollectID}$ stage) and two phases of the $\text{Gather}$ stage (lines 10 and 13). Note that, when $a_i$ executes the $\text{Gather}$ stage, all agents execute the $\text{Gather}$ stage. The $\text{Gather}$ stage interrupts the $\text{CollectID}$ and $\text{MakeGroup}$ stages, but, as described later, the behaviors of the $\text{Gather}$ stage do not affect the behaviors of the $\text{CollectID}$ and $\text{MakeGroup}$ stages if no reliable group exists. Therefore, we do not consider the behaviors of the $\text{Gather}$ stage until a reliable group is created in the $\text{MakeGroup}$ 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, $a_i.state = \text{CorrectID}$ holds. In addition, $a_i$ stores $False$ in variable $a_i.EndCI$ because it has not finished the $\text{CollectID}$ 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. Agent $a_i$ increments $a_i.count$ every round, but this behavior is omitted from the following description.

3.2.1 The $\text{CollectID}$ stage.

Algorithm 2 gives the pseudo-code of the $\text{CollectID}$ stage. In this stage, agents collect IDs of all good agents. The $\text{CollectID}$ stage of $a_i$ consists of $2[\log(a_i.ID)] + 6$ phases. Note that the
Algorithm 2 The \( a_i.x \)-th phase of CollectID stage

1: if \( a_i.x \)-th bit of \( a_i.ID^x = 0 \) then
2: \( a_i.L \leftarrow a_i.L \cup (\text{IDs of agents} \ a_i \ \text{met while waiting}) \)
3: else
4: Explore the network by \( \text{EXPLO}(N) \)
5: \( a_i.L \leftarrow a_i.L \cup (\text{IDs of agents} \ a_i \ \text{met while exploring}) \)
6: end if
7: if \( a_i.x = 2 \lfloor \log a_i.ID \rfloor + 6 \) then
8: \( a_i.\tilde{f} \leftarrow \max\{y \mid (4y+4)(y+1) \leq |a_i.L|\} \)
9: \( a_i.x \leftarrow 1 \)
10: \( a_i.\text{EndCI} \leftarrow \text{True} \)
11: end if

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 \( a_i.ID^x \). If the \( x \)-th bit is 0, \( a_i \) waits for one phase in the \( x \)-th phase (lines 1 to 2 of Algorithm 2). Otherwise, \( a_i \) explores the network by \( \text{EXPLO}(N) \), and then waits for one round in the \( x \)-th phase (lines 4 to 6). During these behaviors, if \( a_i \) finds another agent \( a_j \) on the same node, it records \( a_j.ID \) in \( a_i.L \) (lines 3 and 7).

In the last round of the last phase of the CollectID stage, \( a_i \) calculates the estimated number of Byzantine agents \( \tilde{f} \), that is, \( a_i.\tilde{f} \leftarrow \max\{y \mid (4y+4)(y+1) \leq |a_i.L|\} \) (line 10). As we prove later, \( a_i.\tilde{f} \geq f \) and \( |a_i.\tilde{f} - a_j.\tilde{f}| \leq 1 \) hold for any good agent \( a_j \). Also, \( a_i \) stores True in \( a_i.\text{EndCI} \) (line 12).

3.2.2 The MakeGroup stage.

Algorithm 3 gives the pseudo-code of the MakeGroup stage. In the MakeGroup stage, agents create a reliable group composed of at least \( 4f + 4 \) agents. At the beginning of the MakeGroup stage, if the smallest \( a_i.\tilde{f} + 1 \) IDs in \( a_i.L \) contains \( a_i.ID \), agent \( a_i \) becomes a target agent (i.e., \( a_i.\text{state} \leftarrow \text{TargetAgent} \)) (lines 2 to 3 of Algorithm 3). Otherwise, \( a_i \) becomes a search agent (i.e.,
execute different operations. If a by a
rounds on the current node (lines 8 to 10).

a of agents that a say a
reliable group is created (line 26). First, agent
the estimated number of Byzantine agents among agents on the same node and determines whether
(exploration, a
a candidate contains at least 4
on the current node is at least 4
number of Byzantine agents as follows (line 27). If the number of agents in the
MakeGroup
searches for
a
a
a
a
state
• 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.

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 (For search agents) The ID that the agent searches for.
(For target agents) Its own ID.
F The consensus of f among agents on 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.

Table 2: Variables of agents.

| Variable | Explanation |
|----------|-------------|
| a_i.state ← SearchAgent | (lines 4 to 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. |
| In the first X_N rounds of each phase of the MAKETEAM stage, target agents and search agents execute different operations. If a_i is a target agent, it executes a_i.target ← a_i.ID and waits for X_N rounds on the current node (lines 8 to 10).

Let us consider the case that a_i is a search agent. Search agent a_i uses variable a_i.BL to store IDs of agents that a_i regards as Byzantine agents (initially a_i.BL ← ∅). In the first round of each phase, a_i executes a_i.target ← min(a_i.L \ a_i.BL). After that, a_i searches for the agent with ID a_i.target, say a_i.target, by executing EXPLO(N) (lines 13 to 14). If a_i finds a_i.target on the same node during the exploration, a_i ends EXPLO(N) and waits on the node until the end of the X_N-th round of the phase (lines 15 to 16). We can show that, if a_i.target is good, a_i.target keeps waiting as a waiting agent, and consequently, a_i finds a_i.target and waits with a_i.target. Hence, if one of the following conditions holds, a_i regards a_i.target as a Byzantine agent: (1) a_i did not find a_i.target during the exploration, (2) a_i.target moved to another node while a_i was waiting on the same node, or (3) a_i.target.target ≠ a_i.target.ID holds (lines 18 to 22). In this case, a_i executes a_i.BL ← a_i.BL ∪ {a_i.target.ID}, and so a_i never searches for a_i.target in the later phases of the MAKEGROUP stage (line 23).

In the (X_N + 1)-th round of each phase of the MAKEGROUP stage, a_i computes the consensus of the estimated number of Byzantine agents among agents on the same node and determines whether a reliable group is created (line 26). First, agent a_i calculates the consensus a_i.F of the estimated number of Byzantine agents as follows (line 27). If the number of agents in the MAKEGROUP stage on the current node is at least 4 · a_i.f, agent a_i checks values of f of all agents on the current node and assigns the most frequent value to a_i.F. At this time, if multiple values are the most frequent, a_i chooses the smallest one.

Second, a_i determines whether a reliable group is created (lines 28 to 30). If a target agent a_i.target with a_i.target.target = a_i.target.ID and at least one search agent whose target is a_i.target.ID exists on the same node, the set of the target agent and the search agents is called a group candidate. When a_i belongs to a group candidate, a_i regards the group candidate as a reliable group if the group candidate contains at least 4 · a_i.F + 4 agents. If a_i understands that it is in a reliable group, a_i stores a_i.target.ID in variable a_i.GID as the group ID of the reliable group. Note that, as we prove
Algorithm 3 MakeGroup stage

1: if $a_i.x = 1$ then
2: if the smallest $a_i.\tilde{f} + 1$ IDs in $a_i.L$ contains $a_i.ID$ then
3: $a_i.state \leftarrow \text{TargetAgent}$
4: else
5: $a_i.state \leftarrow \text{SearchAgent}$
6: end if
7: end if
8: if $a_i.state = \text{TargetAgent}$ then
9: // $a_i$ is a target agent
10: Wait for one phase on the current node
11: else
12: // $a_i$ is a search agent
13: $a_i.target \leftarrow \min(a_i.L \setminus a_i.BL)$
14: Search for an agent $a_{\text{target}}$ with ID $a_i.target$ by $\text{EXPLO}(N)$
15: if $a_{\text{target}}$ is on the same node during $\text{EXPLO}(N)$ then
16: Stop $\text{EXPLO}(N)$ and wait until the end of the phase
17: end if
18: if $a_i$ finds that $a_{\text{target}}$ is Byzantine then
19: // This is true if one of the following conditions holds
20: // (1) $a_i$ did not find $a_{\text{target}}$ during the exploration
21: // (2) $a_{\text{target}}$ moved to another node while $a_i$ was waiting on the same node
22: // (3) $a_{\text{target}}.target \neq a_{\text{target}}.ID$ holds
23: $a_i.BL \leftarrow a_i.BL \cup \{a_i.target\}$
24: end if
25: end if
26: if the number of agents in the MakeGroup stage on the current node is at least $4 \cdot a_i.\tilde{f}$ then
27: $a_i.F \leftarrow$ the most frequent value of $\tilde{f}$ of agents that exist on the same node
28: Let $GC$ be a set of agents such that they are on the same node as $a_i$ and their target is $a_{\text{target}}$
29: if $|GC| \geq 4 \cdot a_i.F + 4$ holds and there exists $a_{\text{target}}$ with $a_{\text{target}}.target = a_{\text{target}}.ID = a_i.target$ then
30: $a_i.GID \leftarrow a_{\text{target}}.ID$
31: if the $2 \cdot a_i.F + 2$ smallest IDs in $GC$ contains $a_i.ID$ then
32: $a_i.state \leftarrow \text{ExploringGroup}$
33: else
34: $a_i.state \leftarrow \text{WaitingGroup}$
35: end if
36: end if
37: end if

Later, all other good agents in the reliable group also understand that they are in the reliable group, and assign $a_{\text{target}}.ID$ to their variable $GID$ at the same round. Therefore, agents can identify members of a reliable group by observing variable $GID$. When a reliable group is created, the group is divided into two groups, a (reliable) exploring group and a (reliable) waiting group, to be used in the Gather stage (line s 31 to 34). If the $2 \cdot a_i.F + 2$ smallest IDs among agents in $a_i$’s reliable group contains $a_i.ID$, $a_i$ belongs to an exploring group (i.e., $a_i.state \leftarrow \text{ExploringGroup}$); otherwise, it belongs to a waiting group (i.e., $a_i.state \leftarrow \text{WaitingGroup}$). Note that each of an exploring group and a waiting group contains at least $2 \cdot a_i.F + 2$ agents.
Algorithm 4 GATHER stage

1: if $a_i.state = CorrectID$ then
2:   Wait for two phases on the current node
3: else
4:   // The first phase
5:   if $a_i.state = WaitingGroup$ then
6:     $a_i.GL \leftarrow \{a_i.GID\}$
7:     Wait for one phase on the current node
8:     $a_i.GL \leftarrow a_i.GL \cup \text{(group IDs of exploring groups $a_i$ met while waiting)}$
9:   else
10:      if $a_i.state = ExploringGroup$ then
11:         $a_i.GL \leftarrow \{a_i.GID\}$
12:      else
13:         $a_i.GL \leftarrow \emptyset$
14:      end if
15:     Explore the network by $\text{EXPLO}(N)$
16:     Wait for one round on the current node
17:     $a_i.GL \leftarrow a_i.GL \cup \text{(group IDs of waiting groups $a_i$ met while exploring)}$
18:   end if
19:   // The second phase
20:   if $a_i.GL = \emptyset$ then
21:     Wait for one phase on the current node
22:   else if $a_i.state = WaitingGroup \land a_i.GID = \min(a_i.GL)$ then
23:     Terminate the algorithm
24:   else
25:     Search for a reliable waiting group with group ID $\min(a_i.GL)$ by $\text{EXPLO}(N)$
26:     Terminate the algorithm on the node where the reliable waiting group with group ID $\min(a_i.GL)$ exists
27:   end if
28: end if

3.2.3 The Gather stage.

Algorithm 4 gives the pseudo-code 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 to 2 of Algorithm 4).

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 5 to 18). To do this, agents in waiting groups keep waiting, and other agents (agents in exploring groups and agents not in reliable groups) explore the network. During this behavior, when an agent meets a reliable waiting or exploring group, it records the group ID. After that, in the second phase, they gather on the node where the reliable group with the smallest group ID exists (lines 20 to 27).

Here, we explain how agents determine that other exploring or waiting groups are reliable. Assume that agent $a_i$ finds the exploring or waiting group that good agent $a_j$ belongs to. Recall that the exploring or waiting group initially contains at least $2 \cdot a_j.F + 2$ agents. From this fact, even if $f \leq a_j.F$ Byzantine agents leave the group, $a_j.F + 2$ good agents remain. Consequently,
when \( a_i \) finds the group, \( a_i \) can determine that at least one good agent exists in this group because \( |a_i.\tilde{f} - a_i.f| \leq 1 \) holds. 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, it determines that the group is reliable.

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 reliable waiting group, \( a_i \) collects group IDs of reliable groups in variable \( a_i.GL \) (initially \( a_i.GL \leftarrow \{a_i.GID\} \)) by waiting and observing visiting groups. That is, \( a_i \) waits for one phase, and if \( a_i \) finds a reliable exploring group with group ID \( GID \) while waiting, it executes \( a_i.GL \leftarrow a_i.GL \cup \{GID\} \) (lines 6 to 8). If agent \( a_i \) belongs to a reliable exploring group or does not belong to a reliable group, \( a_i \) collects group IDs of reliable groups in variable \( a_i.GL \) by exploring the network. Initially \( a_i \) executes \( a_i.GL \leftarrow \{a_i.GID\} \) (resp., \( a_i.GL \leftarrow \emptyset \)) if \( a_i \) belongs to a reliable exploring group (resp., does not belong to a reliable group) (lines 10 to 13). After that, \( a_i \) explores the network by \( EXPLO(N) \), and then waits for one round. If \( a_i \) finds a reliable waiting group with group ID \( GID \) during the exploration, it executes \( a_i.GL \leftarrow a_i.GL \cup \{GID\} \) (lines 15 to 17).

In the second phase, all agents gather on the node where the reliable group with the smallest group ID exists. Note that, if no reliable group exists, agents just wait for one phase (lines 20 to 21). Now assume that a reliable group exists. If \( a_i \) belongs to a reliable waiting group and \( a_i.GID = \min(a_i.GL) \) holds (i.e., \( a_i \) belongs to a reliable waiting group with the smallest group ID), it terminates the algorithm (lines 22 to 23). Otherwise, \( a_i \) searches for the reliable waiting group with group ID \( \min(a_i.GL) \) by \( EXPLO(N) \), and terminates on the same node as the group (lines 25 to 26).

### 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, \( a_i \) 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. Hence, the lemma holds.

**Lemma 3.2.** After agents finish the CollectID stage, the followings hold for the estimated number of Byzantine agents: (1) For any good agent \( a_i \), \( a_i.\tilde{f} \geq f \) and \( k \geq (4a_i.\tilde{f} + 4)(a_i.\tilde{f} + 1) \) hold, and (2) For any two good agents \( a_i \) and \( a_j \), \( |a_i.\tilde{f} - a_j.\tilde{f}| \leq 1 \) holds.

**Proof.** First, we prove proposition (1). By Lemma 3.1, \( a_i \) contains IDs of all good agents in \( a_i.L \) at the end of CollectID stage, and so \( |a_i.L| \geq (4\tilde{f} + 4)(\tilde{f} + 1) \) holds. Therefore, we have \( a_i.\tilde{f} = \max\{y \mid (4y + 4)(y + 1) \leq |a_i.L|\} \geq \max\{y \mid (4y + 4)(y + 1) \leq (4\tilde{f} + 4)(\tilde{f} + 1)\} = f \). Also, by the algorithm, we clearly have \( k \geq (4a_i.\tilde{f} + 4)(a_i.\tilde{f} + 1) \).

Next, we prove proposition (2) by contradiction. Let us assume that proposition (2) does not hold. 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 \( a_j.\tilde{f} < p + 1 \), and we have \((4(p + 2) + 4)((p + 2) + 1) \leq |a_j.L| \) by \( a_j.\tilde{f} \geq p + 2 \). Therefore, \( |a_i.L| - |a_j.L| > 8p + 20 > f \) (\( p \geq f \) by proposition (1)) holds. On the other hand, since \( a_i.L \) and \( a_j.L \) include IDs of all good agents, we have \( |a_i.L| - |a_j.L| \leq f \), which contradicts the assumption.

Hence, the lemma holds.

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

**Lemma 3.3.** The followings hold in the MakeTeam stage: (1) \( a_{\text{min}} \) is a target agent, and (2) The number of good target agents is at most \( \tilde{f}_{\text{max}} + 1 \).
Lemma 3.4. 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 $a_i$ updates $a_i.BL$ in a phase only when one of the following conditions holds, and then $a_i$ adds $a_i.target$ to $a_i.BL$. Let $a_{\text{target}}$ be the agent such that $a_i.target = a_{\text{target}}.ID$ holds.

1. Agent $a_i$ did not find $a_{\text{target}}$ during the phase.
2. Agent $a_i$ found $a_{\text{target}}$, but $a_{\text{target}}$ moved to another node during the phase.
3. Agent $a_i$ found $a_{\text{target}}$, but $a_{\text{target}}.target = a_{\text{target}}.ID$ did not hold.

For the base case, we consider the first phase of the $\text{MAKEGROUP}$ 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 $a_{\text{min}}.ID$ or an ID of a Byzantine agent. Since only $a_i.target$ can be added to $a_i.BL$, it is sufficient to consider the case of $a_i.target = a_{\text{min}}.ID$. Since $a_{\text{min}}$ is a target agent by Lemma 3.3 and $a_{\text{min}}$ starts the $\text{MAKEGROUP}$ stage no later than $a_i$, the above conditions to update $a_i.BL$ are not satisfied. Hence, in this case, $a_i$ does not update $a_i.BL$. Therefore, 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 $\text{MAKEGROUP}$ stage of $a_i$. We consider $(t+1)$-th phase of the $\text{MAKEGROUP}$ 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 $a_{\text{min}}.ID$ 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.

Lemma 3.5. When good agent $a_i$ executes $a_i.F \leftarrow \tilde{f}$, there exists good agent $a_j$ with $a_j.\tilde{f} = \tilde{f}$

Proof. Assume that $a_i$ executes $a_i.F \leftarrow \tilde{f}$ on node $v$ in round $r$. By the algorithm, in round $r$, there exist at least $4 \cdot a_i.F$ agents that execute the $\text{MAKEGROUP}$ stage on node $v$. Since $a_i.F \geq f$ holds by Lemma 3.2 there exist at least $4 \cdot a_i.\tilde{f} - f \geq 4f - f = 3f$ good agents that execute the $\text{MAKEGROUP}$ stage on $v$ in round $r$. Also, since variable $\tilde{f}$ of good agents takes at most two possible values by Lemma 3.2 at least $\lceil 3f/2 \rceil > f$ good agents on $v$ 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 3.6. If good agent $a_i$ determines that a reliable group is created on node $v$ in round $r$, there exists a set $A'$ of agents that satisfies the following conditions:

- Set $A'$ contains at least $4 \cdot a_i.F + 4$ agents.
- Good agents in $A'$ determine that a reliable group is created on $v$ in round $r$.
- For any good agent $a_j$ in $A'$, $a_j.F = a_i.F$ and $a_j.GID = a_i.GID$ hold at the end of round $r$.

Proof. Assume that good agent $a_i$ determines that a reliable team is created on $v$ in round $r$. Let $A'$ be a set of agents such that, if $a_j \in A'$ holds, $a_j$ stays on $v$ in round $r$ and $a_j.target = a_i.target$ holds. We prove that $A'$ satisfies the conditions of the lemma. Since $a_i$ determines that a reliable team is created, $A'$ contains at least $4 \cdot a_i.F + 4$ agents. Also, $A'$ contains agent $a_{\text{target}}$ with
Fix an agent \( a_j \in A' \). By Lemmas 3.2 and 3.5, \( a_j \bar{f} \leq a_j.F + 1 \) holds, and hence \( 4 \cdot a_j.F + 4 \geq 4 \cdot a_j, \bar{f} \) holds. This implies that the number of agents on \( v \) satisfies the condition that \( a_j \) calculates \( a_j.F \). Since the situation of \( v \) is the same for both \( a_j \) and \( a_j, F = a_j.F \) holds. Since \( a_j.F = a_j,F \) holds and \( a_j \) observes agents in \( A' \), \( a_j \) also determines that a reliable group is created on \( v \) in round \( r \). Since \( a_j \) executes \( a_j.GID \leftarrow a_j.target \) and \( a_j.target = a_j.target \) holds, \( a_j.GID = a_j.GID \) holds. Hence, the lemma holds. \( \square \)

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 latest.

Lemma 3.7. 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, at most \( (4 f_{max} + 2)f' \) 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, when \( a_{last} \) executes the \((x + 1)\)-th phase of the MakeGroup stage \((1 \leq x \leq f')\), at most \( (4 f_{max} + 2)x \) 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 assigns \( Byz_1.ID \) to their variable target in the second phase. For contradiction, assume \( |A_1| > 4 f_{max} + 2 \). 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 end of the first phase. Therefore, at the end of the first phase, agents in \( A_1 \) and \( Byz_1 \) exist on the same node, and the number of agents is at least \( 4 f_{max} + 4 \). This contradicts the assumption since a reliable group is created by the algorithm. Therefore, \( |A_1| \leq 4 f_{max} + 2 \) holds.

For induction step, assume that, in the \((x + 1)\)-th phase \((1 \leq x < f')\), at most \( (4 f_{max} + 2)x \) good agents assign bid \( \in \{ Byz_1.ID, Byz_2.ID, \ldots, Byz_x.ID \} \) to target. Let \( A_x \) be a set of good agents that assign bid \( \in \{ Byz_1.ID, Byz_2.ID, \ldots, Byz_x.ID \} \) to target in the \((x + 2)\)-th phase. For contradiction, assume \( |A_x| > (4 f_{max} + 2)(x + 1) \). 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 \( \in \{ Byz_1.ID, Byz_2.ID, \ldots, Byz_{x+1}.ID \} \) to target in the \((x + 1)\)-th phase. Since good agents monotonically increase target, \( A_x \subseteq B_x \cup C_x \) holds. Since \( |C_x| \leq (4 f_{max} + 2)x \) holds by the assumption of induction, \( |B_x \cap A_x| \geq |A_x| - |C_x| > 4 f_{max} + 2 \) 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 end of the \((x + 1)\)-th phase. Therefore, at the end of the \((x + 1)\)-th phase, agents in \( B_x \cap A_x \) and \( Byz_{x+1} \) exist on the same node and the number of the agents is at least \( 4 f_{max} + 4 \). This contradicts the assumption since a reliable group is created by the algorithm. Therefore, \( |A_x| \leq (4 f_{max} + 2)(x + 1) \) holds.

Hence, the lemma holds. \( \square \)

Lemma 3.8. Before \( a_{last} \) finishes the \((f - 1)\)-th phase of the MakeGroup stage, a reliable group is created.

Proof. Let \( f'(< f) \) be the number of Byzantine agents whose IDs are smaller than \( a_{min}.ID \). By Lemma 3.7, if a reliable group is not created before \( a_{last} \) finishes the \( f' \)-th phase of the MakeGroup stage, at most \( (4 f_{max} + 2)f' \) good agents assign an ID of a Byzantine agent with a smaller ID than \( a_{min} \) to target in the \((f' + 1)\)-th phase. Also, by Lemma 3.3, the number of good target agents is at most \( f_{max} + 1 \). This implies that, in the \((f' + 1)\)-th phase, at least \( (k - f') - (f_{max} + 1) - (4 f_{max} + 2)f' \) good search agents assign \( a_{min}.ID \) to target (because \( a_{min}.ID \) is not in variable BL of agents by Lemma 3.4). Since they can successfully find \( a_{min} \), by Lemma 3.2, at least \( (k - f') - (f_{max} + 1) - (4 f_{max} + 2)f' \geq 4 f_{max} + 3 \) search
agents stay with target agent \( a_{\min} \) at the end of the \((f' + 1)\)-th phase. This implies that they make a reliable group. Hence, the lemma holds.

The following lemma shows that agents can achieve the gathering if at least one reliable group is created and they finish the COLLECTID stage.

**Lemma 3.9.** Assume that, in round \( r \), the first reliable group is created. Let \( G_{\text{min}} \) be the group with the smallest group ID among reliable groups created in round \( r \), and let \( \text{gid}_{\text{min}} \) be the group ID of \( G_{\text{min}} \). Let \( v_{\text{min}} \) be the node where \( G_{\text{min}} \) is created. The following propositions hold: (1) If \( a_i \) has finished the COLLECTID stage before round \( r \), it terminates the algorithm on \( v_{\text{min}} \) during the first two 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 on \( v_{\text{min}} \) in the first two phases of the GATHER stage after it finishes the COLLECTID stage.

**Proof.** First, we prove proposition (1). We focus on the first two phases of the GATHER stage after round \( r \). In the following, we simply write the first (resp., second) phase instead of the first (resp., second) phase of the GATHER stage after round \( r \). By Lemma 3.6, for any good agent \( a_j \) in a reliable group, the group contains at least \( 4 \cdot a_j \cdot F + 4 - f \) good agents, and hence each of the exploring group and the waiting group contains at least \( 2 \cdot a_j \cdot F + 2 - f \geq a_j \cdot F + 2 \) good agents. This also means that, if an exploring or waiting group contains a good agent, it contains at least \( f + 2 \) agents. We consider three cases depending on the status of \( a_i \) at the end of round \( r \).

(Case 1) \( a_i \) belongs to a waiting group of \( G_{\text{min}} \). Agent \( a_i \) keeps waiting on \( v_{\text{min}} \) in the first phase. Clearly, \( a_i \cdot \text{GL} \) contains \( a_i \cdot \text{GID} = \text{gid}_{\text{min}} \). In addition, since \( f < a_i \cdot f + 1 \) holds, \( a_i \) never determines that an exploring group composed of only Byzantine agents is reliable. Hence, in the second phase, \( a_i \) understands that it belongs to the waiting group of \( G_{\text{min}} \) (from \( \min(a_i, \text{GL}) = \text{gid}_{\text{min}} = a_i \cdot \text{GID} \)). Therefore, it terminates the algorithm on \( v_{\text{min}} \) at the beginning of the second phase.

(Case 2) \( a_i \) belongs to an exploring group or does not belong to a reliable group. Let \( a_i \) be a good agent in \( G_{\text{min}} \). Since \( a_i \) executes \( \text{EXPLO}(N) \) in the first phase, it visits \( v_{\text{min}} \) during the first phase and notices that a waiting group on \( v_{\text{min}} \) contains at least \( a_j \cdot F + 2 \) agents. By Lemmas 3.2 and 3.5, since \( a_j \cdot F + 2 \geq a_i \cdot f + 1 \) holds from \( |a_j \cdot F - a_i \cdot f| \leq 1 \), \( a_i \) determines that this waiting group is reliable. Consequently, \( a_i \) records the group ID of this group (i.e., \( \text{gid}_{\text{min}} \)) in \( a_i \cdot \text{GL} \). Since \( f < a_i \cdot f + 1 \) holds, \( a_i \) never determines that a waiting group composed of only Byzantine agents is reliable. Hence, in the second phase, \( a_i \) searches for the waiting group of \( G_{\text{min}} \) (from \( \min(a_i, \text{GL}) = \text{gid}_{\text{min}} \)). From Case 1, the waiting group of \( G_{\text{min}} \) stays on \( v_{\text{min}} \). Therefore, \( a_i \) terminates the algorithm on \( v_{\text{min}} \).

(Case 3) \( a_i \) belongs to a reliable waiting group other than \( G_{\text{min}} \). In the first phase, while \( a_i \) keeps waiting, the exploring group of \( G_{\text{min}} \) visits the node with \( a_i \). Similarly to Case 2, we can show that \( a_i \) adds \( \text{gid}_{\text{min}} \) to \( a_i \cdot \text{GL} \) and searches for the waiting group of \( G_{\text{min}} \) in the second phase. From Case 1, the waiting group of \( G_{\text{min}} \) stays on \( v_{\text{min}} \). Therefore, \( a_i \) terminates the algorithm on \( v_{\text{min}} \).

Next, we prove proposition (2). Assume that, in round \( r' \), \( a_i \) finishes the COLLECTID stage. From proposition (1), the waiting group of \( G_{\text{min}} \) has already terminated on \( v_{\text{min}} \) in round \( r' \). In the first phase of the GATHER stage after round \( r' \), \( a_i \) executes \( \text{EXPLO}(N) \) and visits \( v_{\text{min}} \). Similarly to the proof of proposition (1), \( a_i \) adds \( \text{gid}_{\text{min}} \) to \( a_i \cdot \text{GL} \) and searches for the waiting group of \( G_{\text{min}} \) in the second phase. Therefore \( a_i \) terminates the algorithm on \( v_{\text{min}} \) during the second phase.

Hence, the lemma holds.

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 weak Byzantine agents, and \( \Lambda_{\text{good}} \) be the largest ID among good agents. If the upper bound \( N \) of \( n \) is given to agents and \((4f + 4)(f + 1) \leq k \) holds, the proposed algorithm solves the gathering problem with non-simultaneous termination in at most \( 3(2[\log \Lambda_{\text{good}}] + f + 7)(X_N + 1) \) rounds.

**Proof.** Let \( a_{\text{last}} \) be the good agent that finishes the COLLECTID stage latest, that is, the agent whose ID is \( \Lambda_{\text{good}} \). Since \( a_{\text{last}} \) executes \( 2[\log \Lambda_{\text{good}}] + 6 \) phases of the COLLECTID stage, \( a_{\text{last}} \) finishes the COLLECTID stage in \( 2[\log \Lambda_{\text{good}}] + 6 \cdot 3(X_N + 1) = 3[2[\log \Lambda_{\text{good}}] + 6](X_N + 1) \) rounds. By Lemma 3.8, a reliable group is created before \( a_{\text{last}} \) finishes the \((f + 1)\)-th phase of the
Therefore, agents achieve the gathering in at most 3(2

In this section, we propose an algorithm for the gathering problem with simultaneous termination. By Lemma 3.9, 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 3(2\log \Lambda_{good}) + f + 7) rounds.

4 An algorithm for the gathering problem 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. By the previous algorithm, all good agents gather on a single node but terminate at different times. 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 time.

By Lemma 3.9, 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 phases of the GATHER stage. To do this, we can use the fact that, when good agent \( a \) finishes the COLLECTID stage, \( a, L \) contains IDs of all good agents. That is, \( \max(a, L) \) is the upper bound of IDs of good agents and hence \( a \) 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 these problems, 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.

In this paragraph, we describe the detailed behavior of \( a \) in the algorithm. First, \( a \) executes the previous algorithm until just before it terminates, but it does not terminate. After that, \( a \) waits on the gathering node of the previous algorithm, say \( v \), during the MAKEGROUP and GATHER stages, and regularly checks whether it can terminate. More concretely, \( a \) executes the following operations in the last round of two continuous phases of the GATHER stage. First, \( a \) updates \( a^F \) in the same way as in the MAKEGROUP stage of the previous algorithm. Then, \( a \) checks variable \( L \) of agents on \( v \). Letting \( L_g \) be a set of IDs that at least \( a_i, F + 1 \) agents on \( v \) have in their variable \( L \), \( a_i \) executes \( a_i, ID_{\text{max}} \leftarrow \max(L_g) \). Since a reliable group has already been created, if the agent with ID \( a_i, ID_{\text{max}} \) has finished the COLLECTID stage, \( a_i \) (and all other good agents) can terminate. Hence, \( a_i \) terminates if \( T = 3(2\log(a_i, ID_{\text{max}})) + 6)(N + 1) \) rounds have elapsed from the beginning.

**Theorem 4.1.** Let \( n \) be the number of nodes, \( k \) be the number of agents, \( f \) be the number of Byzantine agents, and \( \Lambda_{\text{alt}} \) be the largest ID among all agents. If the upper bound \( N \) of \( n \) is given to agents and \( (4f + 4)(f + 1) \leq k \) holds, the proposed algorithm solves the gathering problem with simultaneous termination in at most \( 3(2\log \Lambda_{\text{alt}}) + f + 7)(N + 1) \) rounds.

**Proof.** Assume that, in round \( r \), the first reliable group is created. Let \( G_{\text{min}} \) be the group with the smallest group ID among reliable groups created in round \( r \), \( v_{\text{min}} \) be the node where \( G_{\text{min}} \) is created, and \( a_g \) be a good agent in \( G_{\text{min}} \). From Lemma 3.9, all good agents gather on \( v_{\text{min}} \).

We first prove that all agents on \( v_{\text{min}} \) terminate at the same time. Let \( P \) be the first two phases of the GATHER stage after round \( r \), and let \( r' \) be the last round of \( P \). From Lemma 3.9, good agents in \( G_{\text{min}} \) gather on \( v_{\text{min}} \) in round \( r' \) and hence at least 4 \( a_g, F + 4 - f \geq 3f \) good agents exist on \( v_{\text{min}} \) after round \( r' \). Hence, similarly to Lemma 3.5, good agents on \( v_{\text{min}} \) assign \( f \) of some good agent to their variable \( F \) after round \( r' \). This implies that agents on \( v_{\text{min}} \) assign an ID of some agent to their variable \( ID_{\text{max}} \). In addition, since all agents on \( v_{\text{min}} \) observe the same situation, all good agents on \( v_{\text{min}} \) assign the same value to their variable \( F \), and consequently, they compute the
same $ID_{max}$. Since each good agent $a_i$ on $v_{min}$ terminates if $T = 3(2\lfloor \log(a_i.ID_{max}) \rfloor + 6)(X_N + 1)$ rounds have elapsed, all good agents on $v_{min}$ terminate at the same time.

Next, we prove that, when a good agent on $v_{min}$ terminates, all good agents terminate on $v_{min}$ at the same time. We write $ID_{max}$ and $T$ as the values of variables $ID_{max}$ and $T$ of a good agent that terminates the algorithm on $v_{min}$. Note that, since good agent $a_i$ executes $2\lfloor \log(a_i.ID) \rfloor + 6$ phases of the COLLECTID stage and $a_i.ID \leq ID_{max}$ holds, all good agents finish the COLLECTID stage no later than round $(2\lfloor \log ID_{max} \rfloor + 6) \cdot 3(X_N + 1) - 2(X_N + 1) = T - 2(X_N + 1)$. We consider two cases.

- Case that $r' \leq T$ (i.e., $r \leq T - 2(X_N + 1)$) holds. In this case, no later than round $T - 2(X_N + 1)$, all good agents finish the COLLECTID stage, and a reliable group is created. From Lemma 3.9, all good agents move to $v_{min}$ no later than round $T$. Hence, all good agents terminate on $v_{min}$ at the same time in round $T$.

- Case that $r' > T$ (i.e., $r > T - 2(X_N + 1)$). In this case, all good agents finish the COLLECTID stage no later than round $r$. This implies that, from Lemma 3.9, all good agents move to $v_{min}$ no later than round $r'$. Hence, all good agents terminate on $v_{min}$ at the same time in round $r'$.

Lastly, we prove that good agents terminate in at most $3(2\lfloor \log A_{alr} \rfloor + f + 7)(X_N + 1)$ rounds. Similarly to Theorem 3.1, all good agents gather on $v_{min}$ in at most $3(2\lfloor \log A_{good} \rfloor + f + 7)(X_N + 1)$ rounds, where $A_{good}$ is the largest ID among good agents. In addition, since $ID_{max}$ is an ID of some agent, good agents waits until at most $3(2\lfloor \log A_{alr} \rfloor + 6)(X_N + 1)$ rounds have passed. Hence, good agents terminate in at most $3\lfloor \log A_{alr} \rfloor + f + 7)(X_N + 1) = 3(2\lfloor \log A_{alr} \rfloor + f + 7)(X_N + 1)$ rounds. 

\section{Conclusions}

In this paper, we have developed two algorithms that achieve the gathering in weakly Byzantine environments. The proposed algorithms 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, all agents start the algorithm simultaneously, 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 $O((f + |A_{good}|) \cdot X(N))$ rounds, where $|A_{good}|$ 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 $n$ nodes. The second algorithm achieves the gathering with simultaneous termination in $O((f + |A_{alr}|) \cdot X(N))$ rounds, where $|A_{alr}|$ is the length of the largest ID among agents.

As future work, it is interesting to consider the case that agents start the algorithm at different times. It is also interesting to study the trade-off between the time complexity and the ratio of good and Byzantine agents.

\section*{References}

[1] 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, pages 423–454. 2019.

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

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

[4] Sébastien Bouchard, Yoann Dieudonné, and Anissa Lamani. Byzantine gathering in polynomial time. In ICALP, pages 147:1–147:15, 2018.
[5] T.C. Schelling. The Strategy of Conflict. Harvard University Press, 1960.

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

[7] Dariusz R. Kowalski and Adam Malinowski. How to meet in anonymous network. *Theor. Comput. Sci.*, 399(1-2):141–156, 2008.

[8] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In *SODA*, pages 599–608, 2007.

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

[10] Pierre Fraigniaud and Andrzej Pelc. Deterministic rendezvous in trees with little memory. In *DISC*, pages 242–256, 2008.

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

[12] 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.

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

[14] Samuel Guilbault and Andrzej Pelc. Gathering asynchronous oblivious agents with local vision in regular bipartite graphs. *Theor. Comput. Sci.*, 509:86–96, 2013.

[15] Yoann Diendonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM J. Comput.*, 44(3):844–867, 2015.

[16] Evangelos Kranakis, Danny Krizanc, Euripides Markou, Aris Pagourtzis, and Felipe Ramírez. Different speeds suffice for rendezvous of two agents on arbitrary graphs. In *SOFSEM*, pages 79–90, 2017.

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

[18] Evangelos Bampas, Jurek Czyzowicz, Leszek Gąsieniec, David Ilcinkas, and Arnaud Labourel. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *DISC*, pages 297–311, 2010.

[19] Andrew Collins, Jurek Czyzowicz, Leszek Gąsieniec, and Arnaud Labourel. Tell me where I am so I can meet you sooner. In *ICALP*, pages 502–514, 2010.

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

[21] Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue. Gathering of mobile agents in asynchronous byzantine environments with authenticated whiteboards. In *NETYS*, pages 85–99, 2018.

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