# Gathering algorithms for a strong team of mobile agents in weakly Byzantine environments

Jion Hirose[1], Junya Nakamura[2], Fukuhito Ooshita[1], and Michiko Inoue[1]

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

**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 $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$ 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 take the upper bound $N$ of $n$ as input. 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.

## 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,

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 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, and $f$ is the number of Byzantine agents.

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

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. [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 [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 [6, 7, 8, 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 $n$, $\lambda$, 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. [7] and Ta-shma et al. [8] have improved the time complexity and have proposed algorithms to achieve the rendezvous in time independent of $\tau$. In addition, Millar et al. [9] have analyzed the trade-off between the time required for rendezvous and the number of moves. On the other hand, some papers [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. [10, 11] have proposed algorithms for trees, and Czyzowicz et al. [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 [13, 14, 15, 16] for a finite graph and the literature [17, 18, 19] for an infinite graph.

Recently some papers [2, 3, 20, 21, 4] have studied the gathering problem in the presence of Byzantine agents. 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 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 $|\Lambda_{good}|$ has been proposed. The numbers of non-Byzantine agents required for the gathering algorithms are at least one and $f+2$, respectively, and the 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 $|\Lambda_{good}|$ 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 $n$ and $|\Lambda_{good}|$. 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 $\lceil \log \log n \rceil$ 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 $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$, 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. 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 (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

## 2.1 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 connected to node $v$ is locally and uniquely labeled by function $P_v : \{\{v, u\} \mid u \in N_v\} \to \{1, 2, ..., d(v)\}$ 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.

## 2.2 Mobile agents

There are $k$ agents $a_1, a_2, ..., a_k$ in the system. 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.

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 $(S, \delta)$. 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. 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 $v$, 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 on 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 on a single node obtain the same information of states of the agents.

## 2.3 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.

## 2.4 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 (i.e., the first good agent wakes up) to the round in which all the good agents terminate.

### 2.5 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 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 $b_1 b_2 \cdots b_\ell$ be the binary representation of $a_i.ID$, where $\ell = |a_i.ID|$. 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 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. 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 agents 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 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 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 $a_{target}$ on some node, it ends the search and waits on 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.

**Algorithm 1** Procedure Algorithm($N$) for an agent $a_i$ whose $a_i.ID = b_1 b_2 \cdots b_\ell$ where $\ell = |a_i.ID|$

```
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
```

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. To do this, 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, it contains at least $f + 2$ good agents. Therefore, when an 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. 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 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 CollectID stage. Therefore, we design the behaviors of the MakeGroup stage and the Gather stage so that agents can gather even if the estimated values have the difference. 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, the

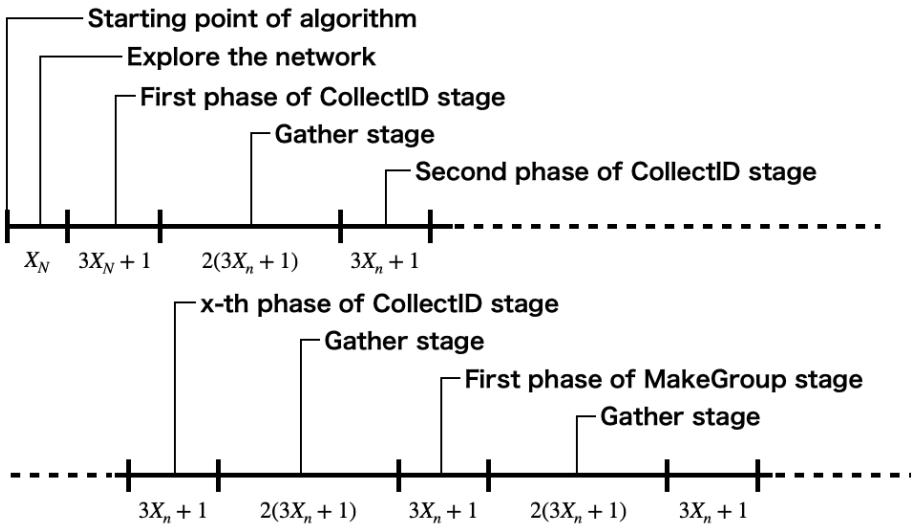


Figure 1: The stage flow.

MakeGroup stage creates a reliable group composed of at least $4f + 4$ agents, 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.

**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). 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.

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. In addition, $a_i$ 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. 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

Table 2: Variables of agents.

| Variable | Explanation |
|---|---|
| *state* | The current state of an agent. This variable takes one of the following values.<br>• *CorrectID* (has not yet finished the CollectID stage)<br>• *SearchAgent* (works as a search agent in the MakeGroup stage)<br>• *TargetAgent* (works as a target agent in the MakeGroup stage)<br>• *ExploringGroup* (belongs to an exploring group in the Gather stage)<br>• *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.<br>Target agents: Its own ID. |
| $F$ | The consensus of $\tilde{f}$ 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. |

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 to 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 $X_N$ round in the $x$-th phase (lines 4 to 7). 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 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, 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).

### 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 composed of at least $4f+4$ agents. At the beginning of the MakeGroup stage, if the smallest $a_i.\tilde{f}+1$ IDs in $a_i.L$ 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 on the current node (line 11). While waiting, $a_i$ executes procedure *consensus*() to determine whether a reliable group is created or not (line 13). We will explain the details of *consensus*() later.

Let us consider the case where $a_i$ is a search agent. The search agent $a_i$ stores in $a_i.BL$ IDs of agents that $a_i$ regards as Byzantine agents (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 $a_{target}$ on the same node during the exploration, $a_i$ ends $EXPLO(N)$ and waits on the node until the end of the phase (lines 21 to 22). We can

**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 on the current node
3: $a_i.L \leftarrow a_i.L \cup \{$IDs of agents $a_i$ met while waiting$\}$
4: **else**
5: Wait for $X_N$ rounds on the current node
6: Explore the network by $EXPLO(N)$
7: Wait for $X_N$ rounds on the current node
8: $a_i.L \leftarrow a_i.L \cup \{$IDs of agents $a_i$ met while exploring$\}$
9: **end if**
10: // The $(3X_N + 1)$-th round
11: **if** $a_i.x = 2\lfloor \log a_i.ID \rfloor + 6$ **then**
12: $a_i.\tilde{f} \leftarrow \max\{y \mid (4y + 4)(y + 1) \leq |a_i.L|\}$
13: $a_i.x \leftarrow 1$
14: $a_i.EndCI \leftarrow True$
15: **end if**
16: Wait for one round

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 to 34), or (2) after $a_i$ finds $a_{target}$, 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 to 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 on the current node is at least $4 \cdot a_i.\tilde{f}$, agent $a_i$ checks values of $\tilde{f}$ of all agents on the current node and assigns the most frequent value to $a_i.F$ (line 2 of Algorithm 4). At this time, if multiple values are the most frequent, $a_i$ chooses the smallest one.

After that, $a_i$ determines whether a reliable group is created. Agent $a_i$ observes states of all agents on 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 $a_{target}.target = a_{target}.ID = a_i.target$, $a_i$ regards the group candidate as a reliable group (line 4). If $a_i$ understands that it is in a reliable group, $a_i$ 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, 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*, for the Gather stage as follows. 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 (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 $2 \cdot a_i.F + 2$ agents.

Once $a_i$ has determined that a reliable group is created, it does not calculate $a_i.F$ and does not determine if a reliable group is created 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 $a_i$ determines 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.

**Algorithm 3** MakeGroup stage for an agent $a_i$

1: **if** $a_i.x = 1$ **then**
2: **if** the smallest $a_i.\tilde{f} + 1$ IDs in $a_i.L$ contain $a_i.ID$ **then**
3: $a_i.state \leftarrow TargetAgent$
4: **else**
5: $a_i.state \leftarrow SearchAgent$
6: **end if**
7: **end if**
8: **if** $a_i.state = TargetAgent$ **then**
9: $//a_i$ is a target agent
10: $a_i.target \leftarrow a_i.ID$
11: Wait for one phase on the current node
12: **and**
13: While waiting, execute *consensus*() every round
14: **else**
15: $//a_i$ is a search agent
16: $a_i.target \leftarrow \min(a_i.L \setminus a_i.BL)$
17: Wait for $X_N$ rounds on the current node
18: Search for an agent $a_{target}$ with ID $a_i.target$ by $EXPLO(N)$
19: **and**
20: **if** meet $a_{target}$ 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 $a_{target}.target \neq a_{target}.ID$ holds
30: $a_i.BL \leftarrow a_i.BL \cup \{a_i.target\}$
31: **end if**
32: **else**
33: // Not meet $a_{target}$ and hence $a_{target}$ is Byzantine
34: $a_i.BL \leftarrow a_i.BL \cup \{a_i.target\}$
35: Wait until the end of the phase
36: **end if**
37: **end if**

### 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 to 2). In the following, we describe the behaviors of agents that have finished the CollectID stage.

If agents have finished the CollectID stage, they try to achieve the gathering in two phases of the Gather stage. In the first phase of the two phases, agents collect group IDs of all reliable

**Algorithm 4** *consensus()* for an agent $a_i$ (Compute the consensus of $\tilde{f}$ and determine whether a reliable group is created)

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

groups (lines 4 to 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 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 16 to 29).

Here, we explain how agents find reliable 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 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 $f \leq a_j.F$ of them are Byzantine, at least $a_j.F + 2$ 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 $|a_i.\tilde{f} - a_j.F| \leq 1$ 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, $a_i$ can correctly recognize the group as 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 reliable 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 to 8). If agent $a_i$ belongs to a reliable 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 to 14).

In the second phase, all agents gather on the node where the reliable group with the smallest group

**Algorithm 5** GATHER stage for an agent $a_i$

1: **if** $a_i.EndCI = False$ **then**
2: Wait for two phases on the current node
3: **else**
4: // The first phase
5: **if** $a_i.state = WaitingGroup$ **then**
6: Wait for one phase on the current node
7: **and**
8: While waiting, whenever $a_i$ meets $a_j$ with $a_j.GID \neq NULL$, execute $a_i.GL \leftarrow a_i.GL \cup \{(a_j.GID, a_j.ID)\}$
9: **else**
10: Wait for $X_N$ rounds on the current node
11: Explore the network by $EXPLO(N)$
12: **and**
13: While exploring, whenever $a_i$ meets $a_j$ with $a_j.GID \neq NULL$, execute $a_i.GL \leftarrow a_i.GL \cup \{(a_j.GID, a_j.ID)\}$
14: Wait for $X_N + 1$ rounds on the current node
15: **end if**
16: // The second phase
17: $//MemberID(gid) = \{id \mid (gid, id) \in a_i.GL\}$
18: $//ReliableGID() = \{gid \mid |MemberID(gid)| \geq a_i.\tilde{f} + 1\}$
19: **if** $ReliableGID() = \emptyset$ **then**
20: Wait for one phase on the current node
21: **else if** $a_i.state = WaitingGroup$ and $a_i.GID = \min(ReliableGID())$ **then**
22: Wait for $3X_N$ rounds on the current node
23: Terminate the algorithm
24: **else**
25: Wait for $X_N$ rounds on 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 on 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**

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 reliable 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 reliable waiting group and satisfies $a_i.GID = \min(ReliableGID())$, it waits for $3X_N$ rounds and terminates the algorithm (lines 21 to 23). Otherwise, $a_i$ waits for $X_N$ rounds, and then, by executing $EXPLO(N)$, searches for the node with the reliable waiting group whose group ID is $\min(ReliableGID())$ (lines 25 to 26). After that, $a_i$ waits until the last round of this phase and terminates the algorithm on the node (lines 27 to 28).

**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 $\Lambda_{good}$ 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 $X_N + 3(2\lfloor \log \Lambda_{good} \rfloor + f + 7)(3X_N + 1)$ 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. By the previous algorithm, all good agents gather on a single node but 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 the algorithm of section 3, 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 on the gathering 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 on the current node. After $a_i$ completes the previous algorithm, it also checks $flag_t$ of all agents on 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 on 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 this paragraph, 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 on 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 on $v$, 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 on $v$ 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 on $v$ 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 on $v$. If all good agents have completed the CollectID stage before round $r_i$, all good agents gather on $v$ 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 on $v$. 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 starts 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 $\Lambda_{all}$ 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 $3X_N + 3(2\lfloor \log \Lambda_{all} \rfloor + f + 7)(3X_N + 1) + 1$ rounds.*

# 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 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] 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 Dieudonné, 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 Gasieniec, 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 Gasieniec, 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.